ENTATION PAGE

Form Approved
OMB No. 0704-0188

imated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

D

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 5 Feb 92	3. REPORT TYPE AND DATES COVERED Research --FY90-92

4. TITLE AND SUBTITLE
Head Tracking and Head Mounted Displays
for Training Simulations

5. FUNDING NUMBERS
$250K

6. AUTHOR(S)
Dr. M. Moshell
Mr. R. Dunn-Roberts
Mr. P. Moskal

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Simulation & Training (IST)
at University of Central Florida (UCF)
12424 Research Parkway, Suite 300
Orlando, FL 32826

8. PERFORMING ORGANIZATION REPORT NUMBER
VSL -
TR92-12

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
PM TRADE (now STRICOM)
12350 Research Parkway
Orlando, FL 32826

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Contr
N61339-90-C-0041

11. SUPPLEMENTARY NOTES
This document is 1 of 2 tasks on the Ref. Contract. A second document "Dynamic Terrain" completes the contract tasking.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unrestricted-Unclassified

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A 2 Part Task.
The first task constructs a 6 monitor display around a Simulated Abrams M1A1 Tank Commander location. The scene is displayed on three monitors at a time and switches to an adjacent three as a function of the head motion sensed from the Tank Commander. Production simulator difficulties were studied.

The second task integrates 'eye phones' and Cyher Face' to various image generators, especially ESIG500 and SIMNET IG. Simulation usefulness issues were researched.

14. SUBJECT TERMS
SIMNET, Head tracking display, work stations, image generators.

15. NUMBER OF PAGES
1

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

INSTITUTE FOR SIMULATION AND TRAINING

Contract Number N61339-90-C-0041
PM TRADE

February 5, 1992

Head Tracking and Head Mounted Displays for Training Simulation

Final Report
Visual Systems Laboratory

iST

Institute for Simulation and Training
12424 Research Parkway, Suite 300
Orlando FL 32826

University of Central Florida
Division of Sponsored Research

IST-TR-92-12

INSTITUTE FOR SIMULATION AND TRAINING

Head Tracking and Head Mounted Displays for Training Simulation

Final Report

The body of this report is VSL Document VSLM92.4. This document is the Final Report of a two year project under Contract N61339-90-C-0041, sponsored by the Army Project Manager for Training Devices (PM TRADE).
All opinions herein expressed are solely those of the authors.

Contract N61339-90-C0041
PM TRADE

February 5, 1992
Visual Systems Laboratory
IST-TR-92-12

Prepared by

Richard Dunn-Roberts, Project Manager ____________________

J. Michael Moshell, Kevin Uliano, Pat Moskal

Reviewed by

Brian Goldiez ____________________

Institute for Simulation and Training • 12424 Research Parkway, Suite 300 • Orlando, Florida 32826
University of Central Florida • Division of Sponsored Research

FINAL REPORT

Head Tracking and Head Mounted Displays for Training Simulation

Richard Dunn-Roberts, Kevin Uliano, Pat Moskal, Michael Moshell

(The Body of this Report is VSL Document 92.4)

Visual Systems Laboratory
Institute for Simulation and Training
University of Central Florida
Orlando, FL 32816

5 Feb 92

Abstract

This document is the Final Report of a two year project at the Institute for Simulation and Training supported by PM-TRADE under contract #N61339-90-C-0041.

This project has two components. The first component is the construction of a six-monitor head-tracking display (HTD) to provide the tank commander (TC) in a SIMNET M1A1 Abrams simulator with a 360-degree panoramic view into the SIMNET database, without requiring additional image generation hardware. This component successfully demonstrated the concept and highlighted some of the difficulties to be overcome in building a production simulator using HTD technology.

The second component is the investigation of low cost head-mounted display (HMD) technology for simulators. This component included attaching HMDs to several available realtime image sources, and describing the difficulties and techniques used to overcome the difficulties. Ultimately, four image sources were used:

- Silicon Graphics Iris workstations;
- An Evans & Sutherland ESIG-500 image generator;
- The SIMNET simulator's image generator; and
- A low-cost (PC-based) Sense8/Intel DVI graphics system.

The two best-known commercial HMDs costing less than $10,000 were evaluated: VPL's *EyePhones* (in two versions) and PopOptix Labs' *CyberFace*. Of the three product versions, only VPL's second-edition EyePhones approached an acceptable level of resolution for training simulation. The devices' limitations, and technical trends, are discussed, with recommendations for further studies.

Contents

I. Introduction

One of the most obvious limitations of low-cost simulation technology, as exemplified by the the SIMNET team training system is the restricted field of view. In a real M1A1 tank, six "vision blocks" (periscopes) provide the Tank Commander (TC) with essentially a complete 360 degree field of view (FOV). Each block spans approximately a 45 degree static horizontal FOV, but the TC can move his head from side to side to view more than 60 horizontal degrees of the surroundings from each vision block.

In the SIMNET system, in order to achieve an acceptable pixel density for target acquisition and navigation, three 320 x 128 visual displays were placed side-by-side to simulate one M1A1 commander's vision block. Each display was set to span 8 degrees (vertical) by 20 degrees (horizontal), thus achieving a 60 degree effective horizontal FOV, and a pixel density of 3.75 arc-minutes per pixel in the horizontal direction.

> SIMNET's vertical pixel density is essentially the same, and henceforth we will discuss only horizontal FOV and pixel densities. The aspect ratio (horizontal to vertical) of 2.5:1 will remain constant throughout these discussions unless noted.

In the healthy human eye, an image centrally presented can generally be resolved into details spanning 0.5 arc minute, and so clearly the SIMNET visual system is far from realistic. However, its designers determined that within the criterion of a 3.5 km maximum range to horizon, 3.75 arc minutes was adequate for target detection when targets were moving against the simplified background of a SIMNET database.

The SIMNET TC turret rotates under the control of a thumb switch on the simulator's machine gun control handle, requiring approximately 12 seconds to rotate 350 degrees. A stop prevents complete rotation (to protect electronic cables from twisting). This can be a severe limitation if the TC needs to look in a direction opposite to the tank cannon.

The central hypothesis of this study was that the tank commander in a SIMNET unit is operating in a handicapped mode, compared to a real tank commander. First, an operational TC usually operates in POP-hatch ("Protected Open Position") mode unless under direct attack or chemical/biological threat conditions. The massive hatch cover is horizontally suspended above the hatch opening to protect from downward fire or fragmentation, but the TC is looking out through a horizontal slit.

No equivalent POP-hatch experience is provided in the SIMNET system.

Second, even when forced to close the hatch, the TC still has six high-quality vision blocks and the ability to see in any direction as rapidly as he can turn his head.

The questions we therefore proposed to study were the following:

> 1. For closed-hatch operation: would a TC's performance in navigation and target acquisition be improved if the experience of having six instantaneously available vision blocks were simulated, instead of requiring the TC to rotate the SIMNET cupola?
>
> 2. For POP-hatch operation: would it be possible to provide visual stimuli, using low-cost head mounted display technology, so that a TC's POP-hatch performance in navigation and target acquisition could be incorporated into the SIMNET experience?

Despite numerous difficulties in working with SIMNET prototypical equipment, some answers to these questions were achieved. We first discuss common technical elements in both projects, namely, the ability to track the subject's head position and orientation. We then discuss the HTD project, which took place mostly in 1990. Finally we discuss the HMD project, which took place in 1990 and 1991.

II. Head Tracking Technology

During the first two months of this task (February-March 1990) we engaged in a literature and product-information search. This search provided us with information that was useful for both projects under this task. In general, this search provided us with information concerning availability and cost of head-tracking equipment and head-mounted displays.

In the realm of head-tracking, at the time of the start of this project, there were only two feasible schemes: magnetic and infra-red. The only IR tracker we came across was used in the CAE-Link head-mounted display. This system was custom-built, and replication would have cost at least $20,000 (our estimate). At the time of the start of this project, the Polhemus magnetic tracker was the only commercially available magnetic tracker, although others have since become available. For this reason, the Polhemus Isotrak was chosen. This is a six degree-of-freedom ten-bit tracking device in wide usage.

The Polhemus product comes in two versions: an economical model for $3,000, and a high-resolution model for $10,000. The economical model tracks spatial positions with a static accuracy of 0.13 inches while source to sensor separation is less than 15 inches, and a static angular accuracy of 0.85 degrees within this range. Data is reported with a resolution of 0.9 inches and 0.35 degrees.

The high resolution model tracks spatial positions with a static accuracy of 0.1 inches, and angular position accuracy of 0.5 degrees. Its resolution of reported data is .046 inches and 0.1 degree, within a 30 inch radius of the emitter. Both models respond at a 30 hz rate. Two sensors can be used with a single emiter and a time-multiplexed System Electronics Unit, thus cutting in half the effective response rate of both sensors.

We selected the economy model upon the recommendations of VPL Research, since it was incorporated with the VPL EyePhones, and it proved adequate. The high resolution tracker is usually used for CAD applications as a manual input device (stylus), where extreme spatial accuracy is needed. The head tracking function is less demanding of positional precision.

Since our purchase of the Polhemus, a new 12-bit magnetic tracking device has become available. This higher resolution device, known as the Bird, is made by Ascension Technology, has some advantages, as well as one disadvantage. The primary advantages are higher resolution and less sensitivity to environmental distortions (metals, EMF, etc.).

The one known disadvantage of the Bird products is a two-foot radius of accuracy, vs. a three-foot radius for the Polhemus. A version of the Bird with an eight-foot radius of accuracy is expected soon from Ascension Technology in 1992, as well as a Flock of Birds that can use up to six sensors with a single source. The Bird products cost around $3000 for a single sensor/emitter pair, equivalent to the economy model from Polhemus. The Flock of Birds uses a single emitter and several (up to six) sensors, each with its own electronics module. Thus no time multiplexing is required and all devices continue to respond at 30 hz. Each additional sensor/electronic setup costs approximately $1500.

In 1992 as the project is ending, a number of other tracking devices are becoming available. Logitec has produced an acoustically based "3d Mouse", which uses a triad of ultrasonic emitters arranged in a triangle approximately 15 cm on a side, and a triad of detectors approximately 10 cm on a side, attached to a three button mouse. Unlike the Polhemus and Bird products, the 3d mouse will not function in all orientations; the sensors must "see" the emitters. This product costs $1000.

III. Head Tracking SIMNET Commander Display (HTD)

The purpose of the HTD is to simulate the M1A1 Abrams tank commander's (TC) cupola, which has six vision blocks. Each vision block has a 45x15 degree field of view (FOV) (approximately) . SIMNET provides a single 60x8 degree FOV using three channels of the image generator.

The HTD design allows us to drive six monitors with the TC's three channels of the SIMNET image generator. This is accomplished through the use of a *video switcher* and the Polhemus magnetic tracker, under control of an IBM PC-AT. The control computer reads the tracker, controls the video switcher, and controls the direction of view (DOV) on the image generator (IG).

We first describe the video switcher and display setup. We then discuss the experiments which were conducted.

A. The Head Tracking Display Testbed

In the HTD, only the three monitors in front of the TC's head are active at any time, with each monitor presenting approximately the same FOV as a real vision block. This gives the TC a 140x16 degree instantaneous FOV. Under control of the PC-AT, the video switcher and the Polhemus tracker activate the three monitors in front of the TC. As the TC turns, the control PC shifts the output signals to the appropriate three monitors. The switcher is designed to give an instantaneous change of view in order to eliminate the time now required for the cupola to rotate mechanically through the same distance. For example, a 180 degree rotation that requires six seconds in the mechanically rotating cupola takes approximately one fifth of a second in the HTD.

The video switcher was designed and constructed by IST. It takes four composite video input signals and routes them to any of six output channels under the control of the control computer. (Figure 1)

The control computer also controls the DOV of the IG through a digital-to-analog converter. This converter mimics the potentiometer in the mechanically rotating cupola to change the DOV of the IG.

The control computer reads the Polhemus magnetic tracker and, based on the orientation of the user's head, sends signal to the IG to update the DOV. The control computer then routes the four input channels to some subset of the six monitors. The routing of input signals to output channels is completely configurable, and any input signal can be routed to any output channel, the only restriction being that only one input signal can be routed to a particular output channel at a given time.

Three of the input channels come from the SIMNET image generator, and the fourth input channel is available for neutral imagery from any video source, or can be left blank.

The control functions of the PC-AT are provided by a combination of hardware additions to the PC-AT and the simulator host and control software on the PC-AT. The hardware additions to the PC-AT consist of a parallel output channel with some logic circuitry and a digital to analog converter. The output channel is used to transmit control signals to the video switcher, and the converter is used to control DOV on the image generator. Both the output channel and the converter reside on a single prototype bus card (called the interface card) in the PC-AT. In addition to this interface card, a game port card has been added to read the synchronization signal from the simulator host.

The modification to the simulator host consists of a small circuit that detects when the simulator A/D card has been read by the host. This circuit sends a signal to the control PC-AT to allow for a measure of synchronization.

The control program for the HTD is fairly simple at a conceptual level. (Figure 2.) It has two basic responsibilities: sending the TC's DOV to the IG, and telling the video switcher which input signals to route to which channel. The control program reads two inputs: the magnetic sensor and the synchronization signal. The only two outputs from the program are a control word for the IG, and a control word for the video switcher.

HTD Electronic Operation:

The outputs from the image generator are in the form of four signals: red, green, blue, and sync. There are three channels: Tank Commander Left (TC-L), Tank Commander Middle (TC-M), and Tank Commander Right (TC_R). To reduce cabling requirements and switcher complexity, the signal is encoded using a Vid I/O video encoder. Output signals from each channel of the IG are connected to the RGB and Sync inputs of a Vid I/O box, as shown below. The connecting cables use BNC connectors. The terminating switches for each channel are set to OFF (vice 75 Ohm) so that the RGB and Sync outputs can be routed to the standard SIMNET monitors. This prevents requirements for recabling when the HTD is not in use.

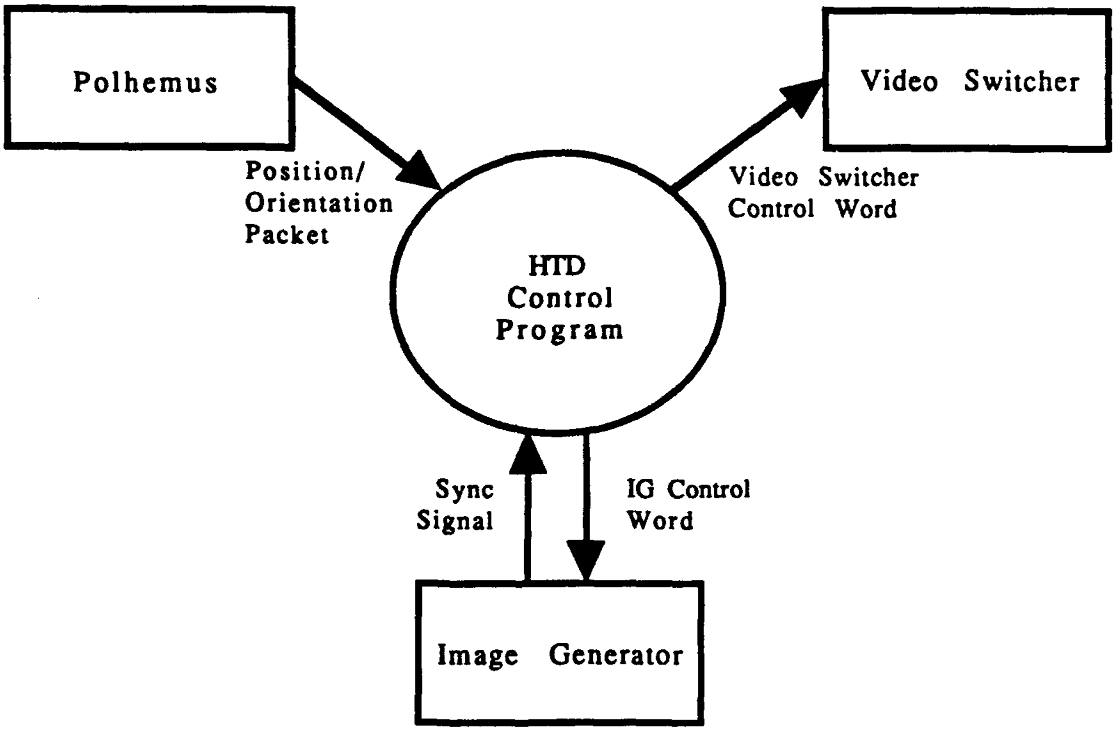


Figure 2. HTD Control Program Block Diagram

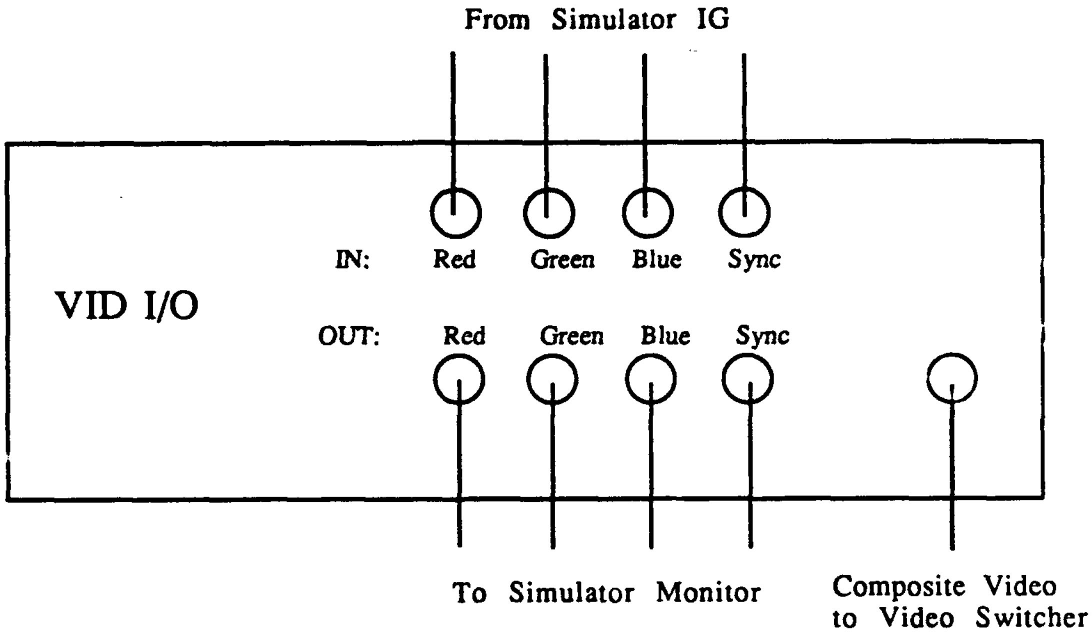


Figure 3. VID I/O Video Encode (x three)

The composite video outputs of the Vid I/O boxes (Figure 3) are connected to the inputs of the video switcher. The input connectors of the video switcher (Figure 4) are labeled 0, 1, 2, and 3. The outputs of the IG are connected to the inputs of the video switcher as follows (with video switcher input channel 3 left open):

IG Output Channel	Video Switcher Input Channel
TC-L	0
TC-M	1
TC-R	2

The outputs of the video switcher are then connected to the color monitors. These outputs are labeled 0, 1, 2, 3, 4, and 5. The video switcher should be switched on prior to connection to the PC in order to prevent damage to the video switching chips.

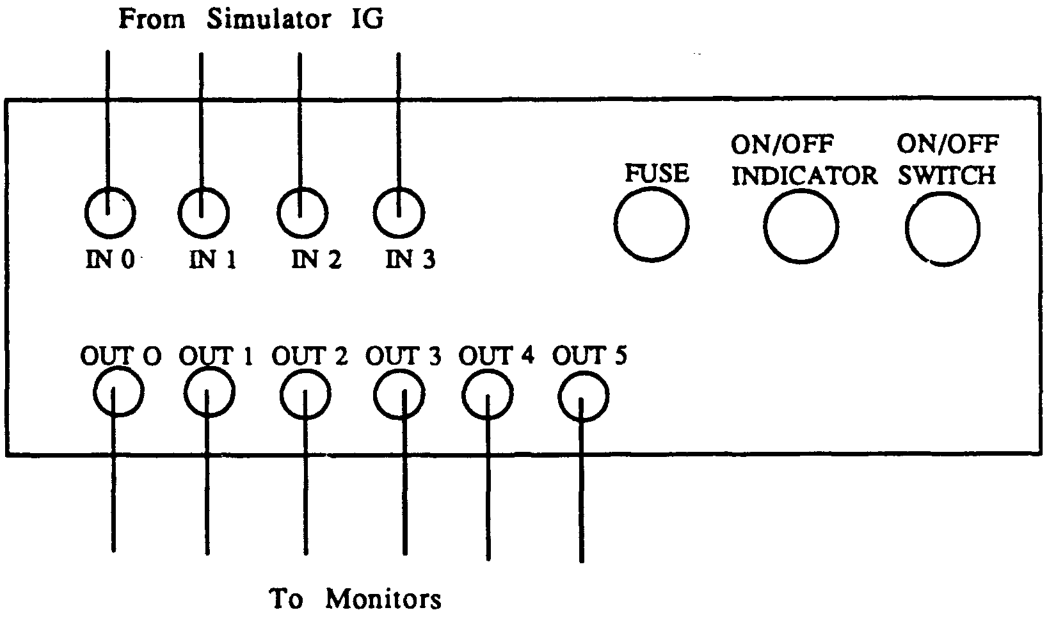


Figure 4. Video Switcher External View

The interface card is connected to the video switcher through a standard six foot DB-25 male to DB-25 male parallel cable. The digital to analog converter is connected to the SIMNET using a cable with a BNC connector at the interface card end and a four pin connector at the SIMNET cupola interface.

The select lines of the video switcher are addressed from the interface card. The interface card has a sixteen bit latch (really two eight bit latches that can be addressed as a single sixteen bit latch) that can be written from the control program using C or assembly language instructions. See figure 5. The base address of the latch is 30Ch. The control program outputs a sixteen bit word to the latch. The first four most significant bits of the address will be ignored, and the twelve low order bits are output to the video switcher.

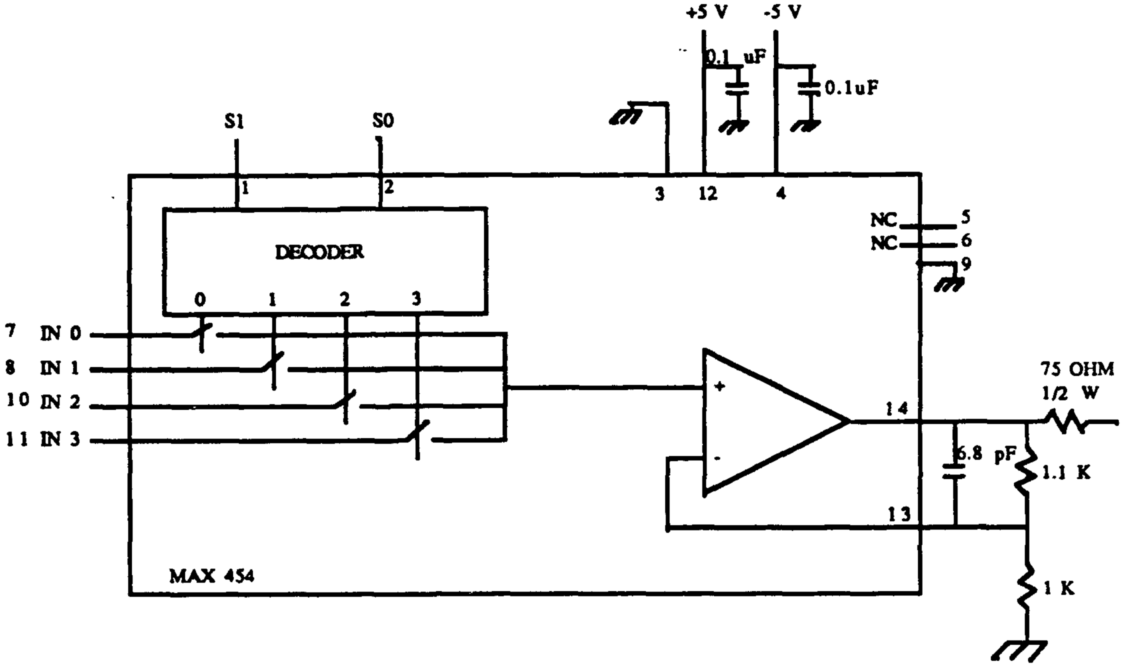


Figure 5: Video Switcher (One module of Six). Functional Diagram

The TC's DOV will be controlled by a digital to analog converter. In its normal operation the field of view is controlled by a potentiometer mounted on the rim of the cupola track. The voltage dropped across the pot varies as the cupola rotates. The voltage drop is converted to a digital value by a A/D converter in the simulator and the DOV is modified accordingly. The HTD bypasses this potentiometer and sends a control voltage from the D/A converter on the interface card to the simulator's A/D converter. The D/A converter is controlled similarly to the video switcher, by outputting a control value to a sixteen bit latch on the interface card. The base address of the D/A converter is 302h.

HTD Physical Configuration:

A simple plywood structure was created to support six Sony 13" video monitors. Standing on six "2 x 4" legs, the hexagonal structure had a sturdy planar top approximately 65" above the floor, with a 30" diameter central hole. A laboratory swivel chair placed on a 6" high platform provided a seat for the subject, whose head projected through the hole in the table into the simulated cupola.

The SIMNET cupola was simulated by a cardboard enclosure. Black poster board was fabricated into a hexagonal chamber 30" in diameter, with six apertures the same size as SIMNET vision blocks. Four sided cardboard cones connected these openings to the Sony TV screens, providing the appropriate fields of view. See Figure 6 below.

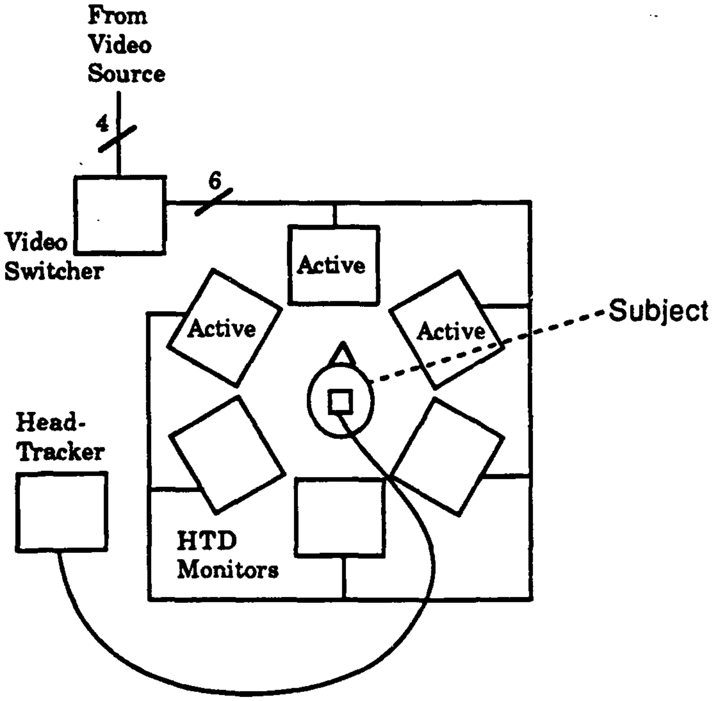


Figure 6: Head Tracking Display Testbed

Because the testbed was in a different room from the SIMNET M1A1 simulator, it was necessary to use walkie-talkies to communicate between experimenters, and the SIMNET intercom for the TC subject to communicate with the driver. These intercoms proved troublesome and unreliable, as has been reported in general with the SIMNET system.

B. Visual Parameters of the Testbed

It was necessary to use three SIMNET channels to drive the three turned-on vision blocks in the HTD system for several reasons. First, these were the only trio of IG channels available to the experimenters. Second, we were comparing a new candidate SIMNET display to the existing displays and thus should be using the same visual databases. Third, a fair comparison should put the same amount of information in front of the subjects in both the experimental and control groups.

However, a SIMNET TC channel was designed to support an 8 x 20 degree field of view. The HTD required that this be re-scaled to span 16 x 40 degrees which represents a 4x increase in angular FOV. Under normal circumstances, this would be expected to overload an image generator's polygon and pixel capacity. The experimenters hoped to "get away" with this situation for the following reasons.

Each SIMNET channel is actually paired with another. Each TC channel is paired with a driver channel, with the TC channel having priority. The combined channel has a polygon capacity of 2000 polygons per frame, at 15 frames per second. SIMNET visual databases are constructed under the assumption of at most 1000 polygons being visible at one time, allocated approximately as follows: 300 terrain polygons, 300 culture (buildings, etc.) and 400 target (tank, Bradley ...) polygons.

Thus, the database was expected to contain at most 300 polygons in any 8 x 20 degree FOV. If a FOV was selected with no culture or targets, even a 4 fold increase in this "worst case" terrain polygon count should have remained within the 2000 polygon capacity of the IG (although we would expect the driver's channels to degrade when the maximum single channel capacity of 1000 polys/sec was exceeded.)

This assumption would only be valid if the view contained few or no culture and targets; or if targets occurred in settings where the polygon density for terrain was well below the maximum design limit. As it turned out, these conditions were often but not always met. Route planning and the search for ways to conceal targets sometimes led to the use of the most complex available terrain, which increased the terrain polygon count.

The SIMNET image generators' performance degraded under these circumstances, producing irregular visual artifacts such as flickering and occasional missing frames. However, a more severe problem with the SIMNET IG's soon manifested itself.

Because of chronic difficulties in accessing the source code for SIMNET during the transition from BBN to Loral's operation of the SIMNET-D site, the project attempted a "black box" approach to SIMNET. That is, we intended to drive the IG/simulator system by stimulating it with signals, but with no modifications to the internal code. Thus, to specify the viewing direction of the TC cupola, an analog driving voltage (described in the previous section) was provided, to emulate the output of the cupola tracking potentiometer.

This voltage changed values whenever the subject's head direction changed. However, the resulting change of IG view direction, and the switching of channels in the video switcher, were unsynchronized, resulting in the unfortunate situation where the subject saw a "pop", or a temporarily incorrect view, on the screen to which he had just turned. An attempt was made to detect the timing of the SIMNET's reading its own A/D converter, but without full access to the scheduling algorithm in the simulator, this external approach could not fully solve the synchronization problem.

Consequently, two classes of visual artifacts occurred in the Head Tracker which were not present in the rotating TC cupola (the control condition for the experiment). As will be seen, these artifacts were reported by the subjects as having a strong influence on their performance.

C. Experiments with Head Tracker

The experiment utilized a within-subjects design, in which each subject receives all conditions in a counterbalanced arrangement. To accomplish the counterbalancing, a Latin Square design is employed, in which each condition immediately preceds and follows the other three conditions once. For example, the first four subjects receive the condition orders shown below in Figure 7.

Subject	Condition Order
1	C1 C2 C3 C4
2	C3 C1 C4 C2
3	C3 C4 C1 C3
4	C4 C3 C2 C1

Figure 7: Latin Square Condition Ordering

The four conditions are:

C1: Terrain Reasoning (Navigation) using Head Tracking Display
C2: Target Acquisition using Head Tracking Display
C3: Terrain Reasoning using the Standard TC Cupola
C4: Target Acquistition using the Standard TC Cupola

Terrain Reasoning Task.

Subjects are shown a 2D paper and pencil map of a section of range at Hunter-Ligget. A start point, an end point, and a general vehicle path are marked on the map, and the Subjects are given two minutes to familiarize themselves with the map before being placed in the M1 simulator or HTD testbed. The subject will then instruct a confederate driver via the SIMNET intercom to drive the stipulated route. The Subject notifies the experimenter when he believes he has arrived at each checkpoint.

Dependent measures include the total time taken to traverse the intended route, and gross location accuracy. The location accuracy is measured by overlaying the Subject's laminated map with marked locations onto the master locator map and measuring the distance from the marked location of the checkpoint to the correct location. The Subject's actual track is also recorded using a simultaneous video/audio recording from a Plan/View

Display, with the audio track containing the Subject's instructions to the driver.

Target Acquisition Task

Subject is shown another 2d paper and pencil map of a range at Hunter Ligget. As in the Terrain Reasoning Task, a start point, an end point, and a general vehicle path are marked on the map, and the Subject is given two minutes to familiarize themselves with the map before being placed in the simulator or testbed. A confederate driver then drives a predetermined and well-practiced route that corresponds to the path outlined on the map. Targets in the form of T-72 tanks are placed along the path at various bearings, distances and orientations.

When the Subject has acquired a target, he will say "Target [right or left]", indicating that the target was located either to the right or left of the vehicle's path of motion. The TC's voice will be recorded and synchronized in time with video taken from the Plan View Display (PVD). The dependent measure for this task will be the number of targets correctly located.

D. Results

Presumably because of the factors described in *Visual Parameters* above, the results reflect no clear preference for either the traditional SIMNET cupola or the HTD. On the terrain reasoning task, two courses were used. On both courses, subjects performed better (by the latency measure) during the early portion of the course (between Checkpoints 1 and 2) with the SIMNET cupola, and better during the period between Checkpoints 2 and 3 with the HTD. This could perhaps be interpreted as saying that subjects were learning how to take advantage of the HTD during the session.

With regard to navigational accuracy, medians show no clear differences. Over Course 1, the cupola showed a less accurate performance; whereas for Course 2, the opposite was observed.

With the target acquisition task, results were similarly unclear. For course 1, the HTD was more successful. For course 2, the cupola was slightly more successful.

Twelve subjects were used. The variance in performance was very large, and no statistical significance can be attributed to any of the above observations. However, the wide variance in itself testifies to the effect of the basic technical problems, on the experiment's ability to measure results.

A subjective evaluation measured preferences and opinions of subjects. In general, they expressed a slight subjective preference for the HTD when asked questions such as "Rate your ability to perceive locations accurately: 1=very easy; 5=impossible". However, the same subjects voted 8 to 4 in favor of the SIMNET cupola when asked "which simulation do you think would be most beneficial for training" and 7 to 5 for the SIMNET cupola when asked "Which simulation do you prefer".

Most significantly, when asked "Do you think your performance would have changed if the head-tracked display did not have popping and flickering?", 11 of the subjects responded YES.

Additional Results. In response to a request from the sponsor, a preliminary concept paper for an advanced version of the Head Tracking Display for POP-hatch operation was developed. This device would have used large monitors to provide an open-hatch display, rather than a simulation of a rotating cupola. This concept paper is attached as Appendix J.

E. Conclusions

1. It is technically possible to build an economical head tracked display device, with video switching to distribute three channels across six displays. The entire prototypical hardware suite cost less than $8,000 in parts, not including the image generators or labor to assemble. A hardened, "simulator ready" version of this equipment would cost perhaps $12,000 in materials, with a fiberglass shell in place of the wooden superstructure.

2. It is technically risky to attempt to use an image generator for any purpose without the full support of its vendor. Artifacts may result which cannot be overcome via purely external means, and which will render experiments difficult or meaningless.

3. The Polhemus magnetic tracking system is reliable and simple to use. However, its latency must be carefully factored into the initial design of the viewing system, as the time required for serial transmission of information is a significant portion of a simulation cycle.

IV. Low Cost Head Mounted Displays (HMD)

A. Head Mounted Display Technology

We looked at various types of head-mounted display technology. HMD's can be opaque (in which only the artificial world is seen) or semi-transparent (in which both artificial imagery and real objects are seen). To date, two main types of HMDs have been constructed.

Pupil-Forming Systems. One type of display is optically implemented as a pupil-forming system[1]. Again, there are two types of pupil forming HMDs. The first uses small (1/2 inch diameter) monochrome CRT's mounted on the side of the helmet. The image is projected through on helmet optics and bounces off a beam-splitter (for semi-transparent operation) or a mirror, into the users eyes. This kind of pupil-forming system costs on the order of $50K and up. Honeywell manufactures this kind of HMD. Monochrome CRT's are usually used to minimize weight.

A variation on these systems uses fiber optics to pipe the image from off-helmet image sources, such as GE light valves. This adds color capability, but is also very expensive. The CAE-Link helmet is of this type.

Infinity-Optics Systems. The other type of HMD uses two small LCD displays mounted directly in front of the user's eyes. Wide angle plastic lenses increase the apparent field of view and provide a virtual image at optical infinity. These systems provide color imagery at a lower cost than the pupil-forming systems but have lower resolution. The NASA VIEW system is a monochrome version of the LCD helmet, and VPL's Eyephones is "nominally" a 442 x 238 pixel color version. This actually represents the number of distinct single-color pixels which are arranged in "triads". The effective resolution if these triads are regarded as single "pixels" is 256 x 137.

The one-eye field of view of the EyePhones is 86 degrees, which yields a horizontal angular resolution of 20 arc minutes per pixel. With a vertical FOV of 76 degrees, the vertical angular resolution is 33 arc minutes per pixel. For details of these computations, please see Appendix I.

VPL has released a high resolution version of the Eyephone with twice the horizontal and vertical resolution, but its cost is approximately $40,000, again making it fairly expensive.

[1] The exit pupil of a pupil-forming optical device is a disc-shaped region in space, to which all of the light from the system converges and from which it diverges. When the eye's pupil is entirely within the exit pupil, the full field of view is perceived at maximum brightness. If the eye's pupil partially overlaps with the exit pupil or is too near or far from the image source, "vignetting" (partial occlusion of the image) occurs, and brightness and clarity diminish.

We are using the medium resolution Eyephones for the HMD Project, and also acquired a variant for the Evans and Sutherland ESIG-500, called Cyberface II by PopOptix Labs (Boston, MA). This firm is owned by Eric Howlett, who produced the lenses for the original VPL Eyephones.

Other Research. There are several on-going low-cost HMD R & D projects going on around the country at this time. One of the most ambitious is the development of micro-laser scanning displays. This project is going on at the University of Washington in Seattle, and, if successful, could provide low-cost, high-resolution, lightweight HMD's in the medium-range future (1993-95).

A second HMD project of interest is underway at IST. Dr. Tom Clarke, with DARPA funding, is constructing an experimental variable-acuity HMD. Using custom electronic hardware, Dr. Clarke's device will pre-distort imagery to concentrate information on the central visual field. A uniform (and thus low-cost) LCD image source will be used. Nonlinear optics will then reverse the pre-distortion, and will result in a varying pixel density in the central and peripheral visual fields. Prototypes should be available in 1993.

B. Driving the EyePhones with Silicon Graphics Workstations

As a first experiment, a testbed was constructed in conjunction with the Dynamic Terrain Project, another PM-TRADE sponsored IST project. A suite of software was constructed which provided stereo displays from two different Silicon Graphics (SGI) workstations, and which was networked to two additional workstations providing models of moving tanks.

It is necessary to set output code in the SGIs to produce NTSC composite video, and to use two Vid I/O boxes to convert the resulting RGB signals to composite. This was accomplished without difficulty, and the Eyephones responded well. The Polhemus controller provided with the EyePhones was interfaced to the SGI via a serial port. This was IST's first experience with Polhemus devices, and served to open the pathway for other uses including the HTD display previously described.

These early VPL Eyephones suffered from a number of problems. On two occasions the devices failed and were returned to the vendor for warranty repairs. The diffuser screens produced a "screen wire" appearance which was quite distracting and had the opposite effect than intended, which was to provide a subjective pixelization of the image. Nevertheless, the first use of the Eyephones was successful in showing that stereo images could be displayed, inter-ocular adjustment provided in software, and a SpaceBall navigation paradigm used with workstation signal sources.

C. Attaching a HMD to the SIMNET System

The original idea was to explore EyePhones as a possible POP-hatch viewing device. Several obstacles presented themselves, as a consequence of the inaccessibility of the SIMNET system's internal details:

1) There was no convenient way to achieve vertical deflection, comparable to the horizontal deflection achieved by the HTD tracker via an emulation of the cupola potentiometer. Instead, the SIMNET IG accepted only the position of a three position switch, to tilt the FOV upward or downward by five degrees from the horizontal.

2) Stereo viewing required the use of two channels of imagery with a horizontal offset, and precise control of the distance between the views. No two SIMNET channels, as originally set up, had these properties. A data block was identified which could respecify fields of view, etc (and was used in the HTD for this purpose).

3) The viewing blocks in SIMNET had a 20:8 aspect ratio, whereas the NTSC signal required for EyePhones required a 4:3 ratio. Only the Stealth configuration of SIMNET would support this viewing situation, and IST's Stealth system had limited hours of availability.

4) Tank commanders often need to look back and forth between the "outside world" and a paper map, but EyePhones did not support this possibility. An alternative would be to treat the EyePhones as binoculars, which could be raised to the face or put down when map viewing was desired.

Meanwhile, IST received from PM-TRADE a supposedly working set of source code for the SIMNET hosts. With the assistance of Warren Katz, one of the software's authors and now a private consultant, IST began to decipher and recompile the SIMNET simulation host source code.

In order to investigate the possibilities, we embarked upon an attempt to integrate the Polhemus system into the SIMNET source code. After some effort with Warren Katz' assistance, we were able to integrate the Polhemus into the Simnet code. We built an internal analog to the external version of the HTD. In essence, we turned off the code which read the cupola potentiometer, and forced in the values from the Polhemus. We managed to extend the Polhemus code to compensate for the "dead spot" where the cupola could not rotate. Thus, full 360 degree rotation about the vertical axis became possible.

The integration of vertical motion was attempted. The problem was traced in the source code back to the way the cupola rotation was stored as an M1 state variable. Only the rotation about the vertical axis is stored. Substantial work would be required to change the source code so as to allow rotation around three axes. Without BBN support, this was not feasible.

We also considered modifying the Stealth system's source code so as to take advantage of its ability to move the viewpoint freely, but found that only about half the Stealth code was actually available. As time and resources were running out, we abandoned further efforts to integrate the EyePhones with the SIMNET system.

D. Attaching a HMD to the ESIG-500 System

The second task attempted under the HMD project was the use of a low-cost head-mounted display on the Evans and Sutherland ESIG-500 image generator. This image generator has several characteristics that made it an attractive platform for this work. First, since the image generator was not tightly coupled with a training simulator, it was hoped that control would be a simpler matter than on the SIMNET, where image generator control code was imbedded inside simulation code. Secondly, it was felt that the higher update rate of the ESIG-500 (50 Hz vice 15 Hz on SIMNET) would provide information on the effects of update rate on users.

The original intention was to use the VPL Eyephones with the ESIG-500. We thought the ESIG-500 was capable of running at a visual system vertical refresh rate (update rate) of 60 Hz. This would allow us to use video encoders to encode the red, green, blue, and sync signals into an NTSC composite signal, which could be used by the Eyephones. Evans and Sutherland told us this would be possible if we did a hardware modification by replacing the timing crystal on the ESIG with a faster crystal, and if we were willing to accept a reduced polygon budget (they said that the update rate times the polygon budget was an invariant, so that as update rate increases, polygon budget decreases). This also would require some microcode patches, which Evans and Sutherland agreed to provide. However, tests showed that the fastest update rate we could achieve was 57 Hz. This was not close enough to the rate required to obtain an image on the Eyephones.

Additionally, we attempted to modify the Eyephones to bring the signal synchronization rate down to 57 Hz. However, it was determined that the cost of producing a crystal that would allow the Eyephones to synchronize at 57 Hz was prohibitive, since it was not a standard crystal and would require a special production run to create. Additionally, it was reported that the circuitry would require substantial modification to allow the Eyephones to synchronize at 50 Hz, even with commercially available crystals.

The next solution we explored was the use of scan converters to modify the ESIG-500 signal from 50 Hz to 60 Hz. Each scan converter could modify one channel and cost $15,000, for a total cost of $30,000. This was clearly a prohibitive cost.

Finally, we were able to locate a different HMD, the Cyberface II, by Pop Optix Labs, that was capable of accepting separate red, green, blue, and synch signals. The Cyberface II is also capable of synchronizing at both 50 Hz or 60 Hz, and therefore can be used directly on the ESIG-500. The ESIG-500 produced a signal with levels inappropriate for the Cyberface II. The images were washed out and detail was difficult to see. These signal strength problems that were solved by circuitry designed and constructed at IST, and incorporated into a housing with the power supply for the Cyberface II. This circuitry provides the ability to adjust each of the red, green, and blue signals individually. This allows us to adjust the brightness and color balance for each display in the Cyberface II HMD system, even where differences in source signals exist. The Cyberface II will be described in more detail later in this report.

Use of a head-mounted display with an synthetic image source such as the ESIG-500 requires control over the image source. With the ESIG-500 this control can be effected through the terminal keyboard attached to the ESIG-500 or through a host computer connected to the ESIG-500 with an Ethernet network. Because it is desirable to use a magnetic tracking device to control eyepoint orientation and, to a lesser degree, eyepoint position, it is necessary to use a host computer controlling the ESIG-500 over an Ethernet connection. The magnetic tracking device is connected to the host computer using a serial connection.

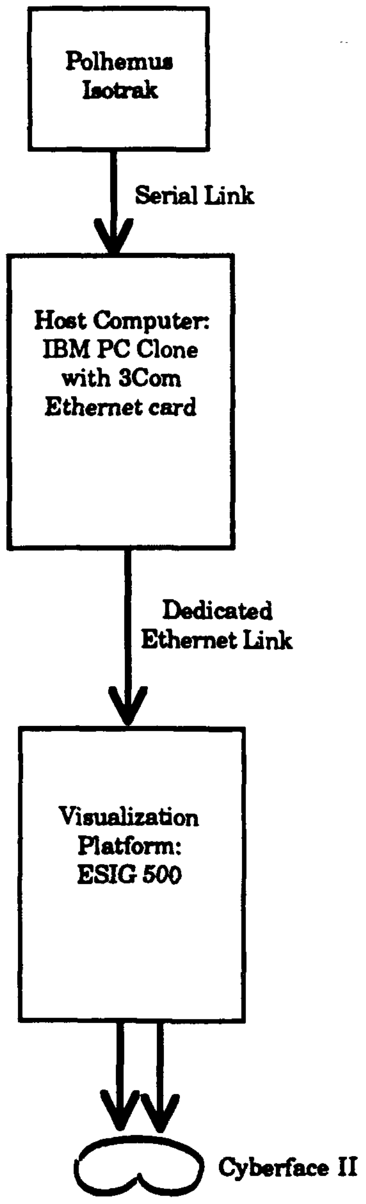

Polhemus Isotrak
Serial Link
Host Computer: IBM PC Clone with 3Com Ethernet card
Dedicated Ethernet Link
Visualization Platform: ESIG 500
Cyberface II

Figure 8: ESIG/CyberFace Configuration

The control program can be summarized by the pseudocode below:

```
initialize serial port
initialize tracker
initialize Ethernet
initialize viewport

while (!done) {
        get a position/orientation record from the Polhemus

        determine the center screen (the one the TC is looking at)
        if (center screen changed) set screen changed flag

        look up the corresponding IG control word
        look up the corresponding video switcher control word

        if (sync signal from IG) {
                if (center screen changed) {
                        send BLANKING control word to the video switcher
                        set delay counter
                        clear screen changed flag
                        set change the switcher flag
                }
                send the IG control word
        }

        if (change the switcher) {
                if (delay counter > 0)
                        decrement delay counter
                else {
                        send video switcher control word
                        clear change the switcher flag
                }
        }
}
```

The ESIG-500 host interface follows IEEE 802.3 hardware and software standards for communication protocol. The communication protocol frame format is shown below:

pream ble	sfd sync	dest addr	src addr	length	data	crc

Figure 9: Frame Format

The preamble, sfd sync, and crc fields are filled in by the Ethernet hardware. The user on the host computer specifies the destination address field, the source address field, the length field, and the data field. The data field contains the command(s) to the ESIG-500 from the host computer. The data field can contain multiple opcodes, with each opcode being followed by any required parameters. The data field must be at least 46 bytes and no more tha:ı 1500 bytes.

The Ethernet standard corresponds to the Physical Layer and the Data Link Layer of the OSI protocol stack. Higher level communications protocols, such as IP and TCP, are not understood by the ESIG-500. For ease of implementation, it was de:ided to use a PC clone as the host computer. This provided easier access to low level Ethernet communication. It also allowed isolation from network traffic (using higher level protocols) not intended for the ESIG-500.

The Etherrnet card used to connect the host PC to the ESIG-500 is a 3Com 503 Ethernet adapter. The host interface library to control the ESIG-500 was originally written in 80x86 assembler using Ethernet adapter control libraries written by the 3Com company and provided with the 503 adapter. The 3Com library was very poorly written. For this reason, for this project, the host interface library was rewritten in the C programming language. The new host interface library uses a public domain 503 adapter control library, also written at IST.

For the purposes of this project, two functions were used from the ESIG-500 host interface library. These two functions are the **escs()** function call and the **esviewpoint()** function call.

The **esviewpoint()** function gives us control over various aspects of the viewport presented on each channel. This allows setting up the image presented to each eye so that the user is able to fuse both images into a three-dimensional image. Different people's eyes have different inter-ocular distances (IOD), and this must be considered when setting up the viewing parameters on each channel. Parameters to this function include the channel number, the x, y, and z coordinates of the eye position, and the heading, pitch, and roll angles of the eye.

These values must be specified for each eye. The **esviewpoint()** function is called twice, once for each eye, during program initialization.

The **escs()** function gives us control over the position and orientation of the eyepoint during run-time. This function is called repeatedly during runtime to continually update the user's point-of-view, based on inp'its from a magnetic head-tracker.

The ISOTRAK magnetic tracker used to sense head position and orientation is connected to a serial port on the host computer. This part of the software is similar to the code written to control the HTD described above.

There are some hardware conflicts that may occur between the serial port and the Ethernet adapter. The PC prioritizes interrupts based on IRQ levels. The Ethernet adapter can be set during initialization to use IRQ levels 2, 3, 4, or 5. The serial ports on a PC typically use IRQ levels 3 or 4. Care must be take to ensure that the IRQ level set for the Ethernet adaptor do not conflict with the IRQ level set for the serial port.

Status of ESIG-500 Testbed

At the conclusion of the project, it is possible to demonstrate a working head mounted display system with the CyberFace display and the ESIG-500. There are still some problems with the head tracking software. However, the principal obstacle to practical use of this system is the poor optical and human-engineering properties of the CyberFace. The display is even fuzzier than the first generation VPL EyePhones, and the head mount is essentially unusable.

> The University of North Carolina Head Mounted Display research team led by Dr. Henry Fuchs has reached similar conclusions regarding this device.

We remain hopeful that we can re-engineer the CyberFace for improved performance, since it is the only presently available device which accepts ESIG signals. A number of experiments are being contemplated, in collaboration with the Army Research Institute, which would require a display device with the rapid update rate and high scene quality of the ESIG-500, together with a more competent display device.

E. Attaching a HMD to the Sense8 System

In January of 1992, an opportunity arose to test the VPL EyePhones with an extremely low-cost image source, to wit an IBM PC containing Intel DVI boards. The PC was provided by an industrially funded IST project; the DVI boards by Dr. Tom Clarke's DARPA-funded IST project. The software, titled *WorldToolKit* from Sense8 Corporation (Sausalito, CA) was donated by Sense8. This system uses a substantial amount of photo-derived texture arranged in 128 x 128 texture maps, which are warped onto polygons in real-time by the DVI boards.

The EyePhones work remarkably well in this context, considering the limitations of the image sources. The image system can produce between one and ten frames per second of imagery, which is similar in speed to the Silicon Grapics Iris demonstrations described above. However, the addition of texture increases the amount of visual content and of visual flow (motion cueing), so that the user's sense of presence is enhanced.

Of the four image sources used with HMD's as part of this project, the Sense8 system is by far the lowest-cost. The entire hardware suite (without EyePhones and Polhemus tracker) cost less than $8,000, and prices continue to drop. The Sense8 software retails for $3,500.

This combination of equipment will be demonstrated at the final presentation of this Project's results in March of 1992.

F. Conclusions

1) With regard to commercially available low cost head mounted display hardware as of the end of 1991, we do not recommend the immediate application of these displays for training purposes. They are neither of sufficiently high resolution, nor physically robust enough for inclusion in training systems.

2) It is likely that the remaining problems of resolution and physical comfort can and will be overcome by a combination of academic and industrial research and development during the next 24 to 36 months. The primary driving force in this market is commercial/entertainment, with a number of new devices appearing on the market as this report is written.

3) The continuing rapid development of low cost image sources such as the Sense8/DVI system will exert an equally strong market force in favor of this technology. Video games of all sorts will appear in 1992 incorporating both low cost HMD's and realtime textured 3d imagery, and this technology should be closely monitored for use in military training.

Appendix A:

Paper presented at SIMTEC '91 Conference

Proceedings of SIMTEC '91
Simulation Computer Society,
Oct. 21-23 1991, Orlando, FL

HEAD-TRACKING DISPLAY DEVICES FOR PANORAMIC VIEWS IN LOW-COST SIMULATORS

Richard Dunn-Roberts, Marty Altman, J. Michael Moshell,
Curtis R. Lisle, Pat Moskal, Kevin Uliano, and Takis Kasparis
Institute for Simulation and Training
and
Electrical Engineering Department
University of Central Florida
Orlando, FL 32816

ABSTRACT

A chronic problem for visual simulation is the requirement for a wide field of view which provides sufficient pixel and object density close to the central viewing axis. In high-fidelity (high cost!) flight simulators with dome displays, high definition area of interest inserts have been used to increase the subject's ability to acquire and track targets.

The authors have designed three display systems to explore low-cost solutions to this problem. These systems have been designed as retro-fits to the SIMNET M1A1 tank simulator. The common problem being addressed is that of a tank commander's view of the world. The three systems are:

- a six-window simulation of the M1A1's vision blocks, to simulate closed-hatch operations;
- a head-mounted display, to simulate protected-open position (POP) hatch operations; and
- a ten-monitor panoramic display, to simulate POP hatch operations without the encumbrance of the head-mounted display.

This paper describes the magnetic sensor technology used to detect the tank commander's viewing direction; the switching technology required to distribute image generator (IG) channels across multiple devices; and the resolution and slew rate requirements and capabilities of the IG used in each design.

A concluding section describes experiments to assess the training effectiveness of the implemented designs with regard to navigation and target acquisition tasks.*

INTRODUCTION

The SIMNET M1A1 tank simulator (hereinafter referred to as SIMNET) is a team-trainer that supports the training of four man teams, including a driver, a loader, a gunner, and a tank commander (TC). The trainees can observe the "world" outside the simulated tank through vision blocks that attempt, more or less, to simulate the periscopes and sights provided for a crew in a real M1A1 tank. There is a total of eight vision blocks; one each for the loader and gunner, and three each for the driver and TC. The image generator used in the SIMNET is a BBN GT101.

In the standard SIMNET, the TC's three vision blocks each provide a 20x8 degree field of view (FOV) into the visual database. These views abut each other, for a total FOV of 60x8 degrees. The direction of view (DOV) is controlled by the TC, and can be mechanically rotated through (just less than) 360 degrees. This represents a low cost solution to the requirement for a wide FOV.

However, because the DOV is mechanically changed, a relatively large delay is introduced when the TC wishes to change the DOV. It takes approximately six seconds to rotate the DOV through 180 degrees. At the Visual Systems Laboratory of the Institute for Simulation and Training (VSL/IST) at the University of Central Florida, the authors have designed three low cost solutions to this problem that change the DOV at electronic speeds. At the time of this report, one of the designs has been implemented and a second in under construction.

All of these systems use magnetic head-tracking technology to sense the direction the TC is looking. Two solutions use this head-tracking information to switch video sources to output channels and update the DOV. The third solution also uses the information to update the DOV, but the TC uses a head-mounted display to view the image, so no video switching is required.

* This work was sponsored by the Army's Project Manager for Training Devices (PM-TRADE).

THE POLHEMUS MAGNETIC TRACKER

Each of the systems described in this paper uses a head-tracking device to determine the position and orientation of the user's head. Head-tracking systems typically either use a magnetic source and sensor, or infra-red diodes and video cameras. Our systems use magnetic tracking technology. Other magnetic trackers are also available, but Polhemus trackers are probably the most commonly used. The Polhemus 3SPACE® Isotrak® is a low cost magnetic tracker and was used in the projects described in this paper.

The Polhemus Isotrak is a six degree-of-freedom measuring device. The Isotrak can provide Cartesian coordinate (x, y, z) and orientation (yaw, pitch, roll) information about a sensor relative to a source positioned near the sensor. The Isotrak will provide this information within a specified accuracy (position - 0.25 inch RMS, orientation - 0.85° RMS) and resolution (position - 0.18 inch RMS, orientation - 0.35° RMS) up to 30 inches away from the source. The host-Isotrak interface is by RS-232C serial link, with user selected baud rates from 300 to 19,200 baud. The information can be in ASCII or binary format, and the highest output update rate is 60 Hz at 19,200 baud in binary format. (Polhemus 1987)

HEAD-TRACKING DISPLAY SYSTEMS

Two of the systems described in the introduction are head-tracking display systems not based on either dome projection systems, which are high cost, or on head-mounted display technology. These two systems, called the Head-Tracking Display (HTD) and the Extended Head-Tracking Display (EHTD), are based on the use of standard video monitors with video switching technology. This allows the use of a number of IG channels with a larger number of monitors to reduce the cost of image generation resources while not reducing the apparent number of output channels.

The VSL/IST Video Switcher

The video switcher used in the Head-Tracking Display systems was designed and built at the Visual Systems Laboratory. The switcher was constructed to take four NTSC composite video input signals and route them to any of six output channels under control of a host computer. It is constructed from six four-to-one composite video multiplexers with two select lines each.

A DB-25 pin connector allows connection of the switcher to the host through a parallel port. Twelve lines of the parallel port are read as a control word to select which input signals are routed to which output channel.

The routing of input signals to output channels is software configurable through the control word. Any input signal can be routed to any output channel, the only restriction being that only one input signal can be routed to a particular output channel at a given time.

The design can be scaled up using additional video multiplexers. It may also be possible to improve signal quality by using RGB video multiplexers instead of composite video multiplexers, but this would increase switcher circuitry complexity and video cabling requirements. (Dunn-Roberts *et al.* 1991)

The Head-Tracking Display

The first implemented display system is the Head-Tracking Display (HTD) (figure 1). This display system has been implemented and is in use with SIMNET.

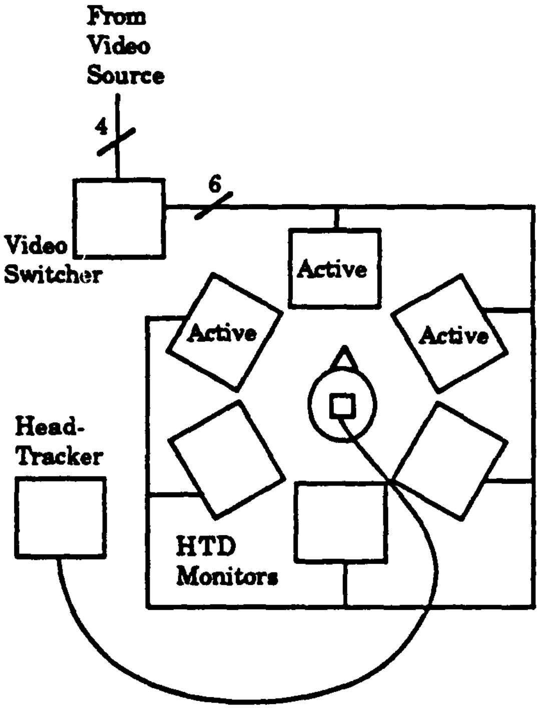


Figure 1. The Head-Tracking Display (HTD)

The purpose of the HTD is to simulate the M1A1 Abrams tank commander's (TC) cupola, which has six vision blocks. Each vision block has a 45x15 degree FOV (approximately). SIMNET provides a single 60x8 degree FOV using three channels from the SIMNET IG. The HTD design allows us to drive six monitors with the TC's three channels of the SIMNET IG. This is accomplished through the use of the video switcher and the Polhemus Isotrak magnetic tracker, under control of an IBM PC-AT. The control computer reads the tracker, controls the video switcher, and controls the direction of view (DOV) on the IG.

In the HTD, six 13" (diagonal) monitors are placed in a ring around the TC's cupola. The cupola is constructed from white cardboard with six openings equally spaced around the TC's head, through which the TC views the images on the monitors. Only the three monitors in front of the TC's head are active at any time, with each monitor presenting approximately the same FOV as a real vision block. This gives the TC a 140x16 degree instantaneous FOV. Under control of the PC-AT, the video switcher activates the three monitors in front of the TC. As the TC turns, the control PC shifts the output signals to the appropriate three monitors. The switcher is designed to give an instantaneous change of view in order to eliminate the time now required for the cupola to rotate mechanically through the same distance. For example, a 180 degree rotation that requires six seconds in the mechanically rotating cupola takes approximately one fifth of a second in the HTD.

The control computer also controls the DOV of the IG through a digital-to-analog (D/A) converter. This converter mimics the potentiometer in the mechanically rotating cupola to change the DOV of the IG.

The control computer reads the Polhemus magnetic tracker and, based on the orientation of the user's head, sends a signal to the IG to update the DOV. The control computer then routes the four input channels to some subset of the six monitors.

Three of the input channels come from the SIMNET IG, and the fourth input channel is available for neutral imagery from any video source, or can be left blank.

The control functions of the PC-AT are provided by a combination of hardware additions to the PC-AT and the simulator host and control software on the PC-AT. The hardware additions to the PC-AT consist of a parallel output channel with some logic circuitry and a D/A converter. The output channel is used to transmit control signals to the video switcher, and the converter is used to control DOV on the IG. Both the output channel and the converter reside on a single prototype bus card (called the interface card) in the PC-AT. In addition to this interface card, a game port card has been added to read a synchronization signal from the simulator host.

The control program for the HTD is conceptually simple. It has two basic responsibilities: sending the TC's DOV to the IG, and telling the video switcher which input signals to route to which channel. The control program reads two inputs: the magnetic sensor and the synchronization signal. The only two outputs from the program are a control voltage for the SIMNET IG, and a control word for the video switcher.

The outputs from the SIMNET IG are in the form of four signals: red, green, blue, and sync. To reduce cabling requirements and switcher complexity, the signals are encoded using a commercial video encoder. Output signals from each channel of the SIMNET IG are connected to the RGB and Sync inputs of the encoders. The composite video outputs of the encoders are connected to the inputs of the video switcher. The outputs of the video switcher are then connected to the color monitors.

The select lines of the video switcher are addressed from the interface card. The interface card has a sixteen bit latch that can be written to by the control program using C or assembly language instructions. The control program writes the control word to the latch, and the twelve low order bits are written to the video switcher.

The TC's DOV is controlled by a D/A converter. In SIMNET normal operation the DOV is controlled by a potentiometer mounted on the rim of the cupola rotation track. The voltage dropped across the pot varies as the cupola rotates. The voltage drop is converted to a digital value by an analog-to-digital (A/D) converter in the simulator and the DOV is modified accordingly. The HTD bypasses this potentiometer and sends a control voltage from the D/A converter on the interface card to the simulator's A/D converter. The D/A converter is controlled similarly to the video switcher, by writing a control value to a sixteen bit latch on the interface card.

The hardware modification to the simulator host consists of a small circuit that detects when the simulator A/D card has been read by the host. This circuit sends a signal to the control PC-AT to allow for a measure of synchronization. However, even with this modification to the SIMNET host, synchronization is not exact, and a blanking interval of approximately one frame is required. (Dunn-Roberts *et al.* 1991)

The Extended Head-Tracking Display

There is significant interest in extending the functionality of the SIMNET M1 trainer to include the Protected Open Position for the tank commander (referred to as the "POP hatch"). The authors have also designed an Extended Head-Tracking Display. This display system has not yet been implemented. The purpose of the EHTD is not only to simulate the M1A1 Abrams TC's cupola, but also to provide for POP hatch operations, while conserving IG channel capacity.

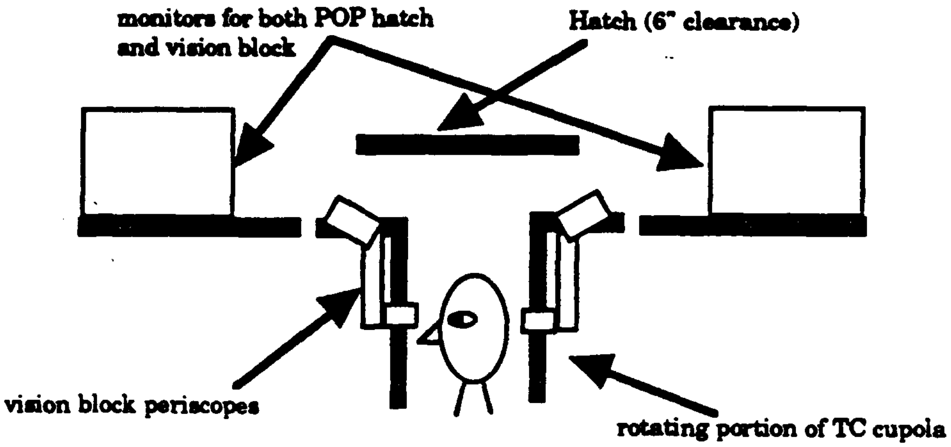

monitors for both POP hatch and vision block
Hatch (6" clearance)
vision block periscopes
rotating portion of TC cupola

Figure 2. The Extended Head-Tracking Display (EHTD), Design 1

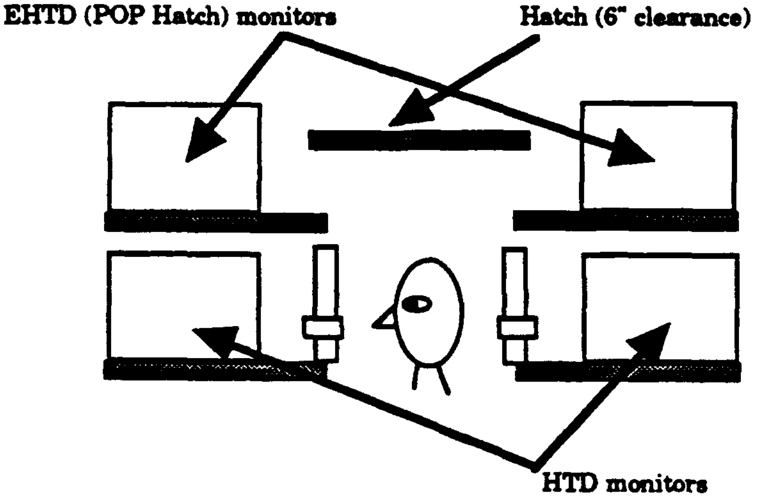


Figure 3. The EHTD, Design 2

Two possible designs have been considered for the EHTD. In both designs, a ring of ten 27" (diagonal) video monitors provides the 360° horizontal FOV POP hatch view In one design, actual periscopes pointing at the ring of monitors provide the view through the TC's vision blocks (figure 2). In the other design, the view through the vision blocks is provided by a second ring of six monitors, just as in the HTD (figure 3).

The EHTD uses the same video switching technology as the HTD. Depending on the design chosen, video signal switching from the IG can be accomplished either with a single switcher, or may require two switchers or a scaled up switcher. This also depends on whether the POP hatch view will activate three or five monitors in the TC's DOV. This question will be addressed with IG requirements later in this paper.

Limitation on requirements for vertical FOV is based on the 6" high opening available under the elevated hatch and the commander's head position relative to the radius of the hatch. Also, the capability for providing high vertical FOV is limited by the aspect ratio of off-the-shelf NTSC monitors.

In the EHTD, the POP hatch view will adjust for head motion within the cupola. Since the commander has a range of available head motion within the cupola (approx. 32" in diameter in SIMNET), the view out the monitors should adjust for the correct head position.

This requires head position information be passed over to the SIMNET host and used to adjust the viewpoint generated by the SIMNET IG. Engineering development will be necessary to provide this level of control over the IG (greater than in the existing SIMNET cupola or in the current HTD).

Visible elements of the tank (the tank hull, the main gun, etc) will need to be displayed in the views from the POP hatch. Since the commander's eyepoint will be above the top of the turret in the POP hatch position, portions of the tank will often be visible. The visible tank features must be modelled or mocked up so they will be displayed correctly.

The cupola will rotate in the EHTD. The current SIMNET allows the commander's cupola to mechanically rotate but provides only a restricted view out of one vision block of the cupola. In a real tank, the cupola is sometimes rotated so as to position the machine gun out of the forward line of sight, or to aim the machine gun. Using an existing SIMNET hull and cupola mechanism, the required vision blocks and a simulated hatch cover and machine gun mount will be incorporated so as to rotate within the panoramic monitor display.

In the EHTD, the user will be able to use the six vision blocks simultaneously with the POP hatch views. The commander will be able to support training in normal mode (through the six vision blocks) or POP hatch (through the out-the-hatch monitors). Both of the designs described earlier for the EHTD support this capability. (Moshell *et al.* 1991).

Image Generator Characteristics

A fundamental goal of the POP hatch display is to allow the platoon commander to acquire targets and to navigate. It would be unacceptable if his visual image were less accurate than those provided through the vision blocks of the SIMNET in its standard configuration.

When the FOV is greatly expanded, two distinct costs are incurred.

- Additional pixels must be supplied so that the visual angle subtended by pixels remains constant; and
- Additional polygon capacity must be supplied so that the greater scene complexity doesn't overload the geometry engine.

To accomplish these goals, the EHTD would require the addition of a BBN GT120 image generator (or equivalent) to the existing SIMNET GT101.

With three screens active simultaneously, the horizontal FOV would be 108°. If five screens are live, the active area would be 180°. (Moshell *et al.* 1991)

Three Active Screens The GT120 can provide two medium resolution channels of 320 x 240 pixels with 3000 polygons, and one high resolution channel of 640 by 480 pixels with 6000 pixels.

This will provide a central channel whose pixels subtend 3.4 minutes of arc (compared to 3.75 minutes in SIMNET), with a polygon density of 6.5 polygons per square degree (compared to 6.25 in SIMNET). Thus, the central channel will slightly improve upon SIMNET's scene and pixel density capacities.

The two adjacent channels will have pixels which subtend about 6.5 minutes of arc (essentially what one sees on the SIMNET Stealth displays) with a polygon density of 3.3 polygons per square degree. These channels will be somewhat more susceptible to overloading on complex scenery than the central channel. (Moshell *et al.* 1991)

Five Active Screens Optionally, the SIMNET could sacrifice some channels from its original IG so that two additional adjacent channels could be rendered. The pixel density is not bad - in fact, at 320 x 256, it is somewhat better than the medium-resolution channels of the GT120.

However, these channels can support only 2000 polygons. With an angular density of only 2.2 polygons per square degree, these channels would quite often overload.

To provide these two channels would require consuming four of the eight SIMNET channels, presumably leaving one each for the the gunner and loader and two channels for the driver. The horizontal field of view for the driver's two channels could be expanded to equal the total FOV of the original three channels. (Johnston 1987; Moshell *et al.* 1991)

HEAD-MOUNTED DISPLAY SYSTEM

We are also designing a third system that utilizes head-tracking information to control the visual display system. This system would use a color LCD head-mounted display (HMD) with a SIMNET IG to present the POP hatch view to the TC.

The first design problem is the interfacing of the magnetic head-tracker with the SIMNET IG. As described above, the HTD uses a voltage to control the TC's DOV. This allows control of the yaw of the DOV, but not the pitch or roll. To allow the TC six degrees-of-freedom with the HMD, the SIMNET host will have to be modified to change the DOV based on the position and orientation of the TC's head as read by the head-tracker. This will require greater control over the the SIMNET host software than is required in either the HTD or the EHTD.

In addition, the HTD uses the existing SIMNET IG to produce three half-height video images. To use the HMD with the M1A1 simulator would require hardware modifications to produce two full-height video channels. Optionally, IST/VSL has a SIMNET Stealth Vehicle simulator that produces three full-height video images that may be used to drive the HMD.

Another challenge is that human eye does not have constant characteristics over the entire visual field. For an image to be as realistic as possible, some predistortion needs to be done to the image before it is presented in a conventional HMD. At IST, this problem is being addressed in a separate project to construct a HMD that more closely matches the image perceived by a human eye. (Clarke 1990)

Perhaps the most difficult problem is how to present an image of the inside of the tank, complete with working controls. An alternative is to mount the HMD in such a fashion as to allow the TC to look through the HMD to see the POP hatch view, and to remove the HMD to work inside the tank cupola. Still, modeling of the exterior of the tank will be necessary to present the TC with a "realistic" view out of the POP hatch.

This system is currently under design and will be implemented in early Fall, 1991. (Altman, Moshell, and Dunn-Roberts, 1991)

EXPERIMENTAL EVALUATION OF SYSTEMS

We are currently evaluating the HTD system on two tasks: terrain-reasoning and target acquisition. As each of the other systems gets implemented, we will use the same experimental design to evaluate them.

In the terrain-reasoning task, tank commanders observe the terrain database while they are driven through a specified geographic area. Their task is to locate three checkpoints specified on an available map. The TCs give the driver directional and speed commands. They tell the driver to stop when they think that they have arrived at each checkpoint. Dependent measures are speed and accuracy of position identification.

In the target acquisition task, TCs scan for targets as they are again driven through a specified course in the database. TCs do not give directional or speed commands for this task. Dependent measures are number of correct target identifications and target acquisition speeds.

Trained drivers are used for all conditions. A common terrain database, representative of the Ft. Knox range, is used for all displays. We are also collecting preference and "wellness" data (e.g. nausea, eye strain) concerning each cupola simulation. We are using both experienced and novice subjects as tank commanders. Each subject participates in both tasks, target acquisition and terrain-reasoning. Each subject will also use both the SIMNET mechanically rotating cupola as well as the HTD. Counterbalancing of task and cupola conditions is employed to remove the potential for confounding effects.

Results of the evaluation of the HTD will be available in August, 1991.

REFERENCES

Altman, M.; J. M. Moshell, R. Dunn-Roberts. 1991. "Technical Considerations for Use of Head-Mounted Displays with SIMNET." VSL Memo 91.18. Visual Systems Laboratory, IST, Orlando, FL (June).

Clarke, T. 1990. "Optimal Virtual World Displays." DARPA BAA #90-15 Proposal. IST, Orlando, FL (Oct.).

Dunn-Roberts, R.; R. DaSilva; and M. Altman. 1991. "First Year Report, BAA Contract #0041, Visual Display Technology R & D." Visual Systems Laboratory, IST, Orlando, FL (Apr.).

Johnston, R. S. 1987. "The SIMNET Visual System." *Proceedings of the 9th ITEC Conference.* Washington, D. C. (Nov.).

Moshell, J. M.; C. Lisle; R. Dunn-Roberts; E. Smart. 1991. "Extending the SIMNET Head-Tracking Display." VSL Memo 91.6. Visual Systems Laboratory, IST, Orlando, FL (Feb.).

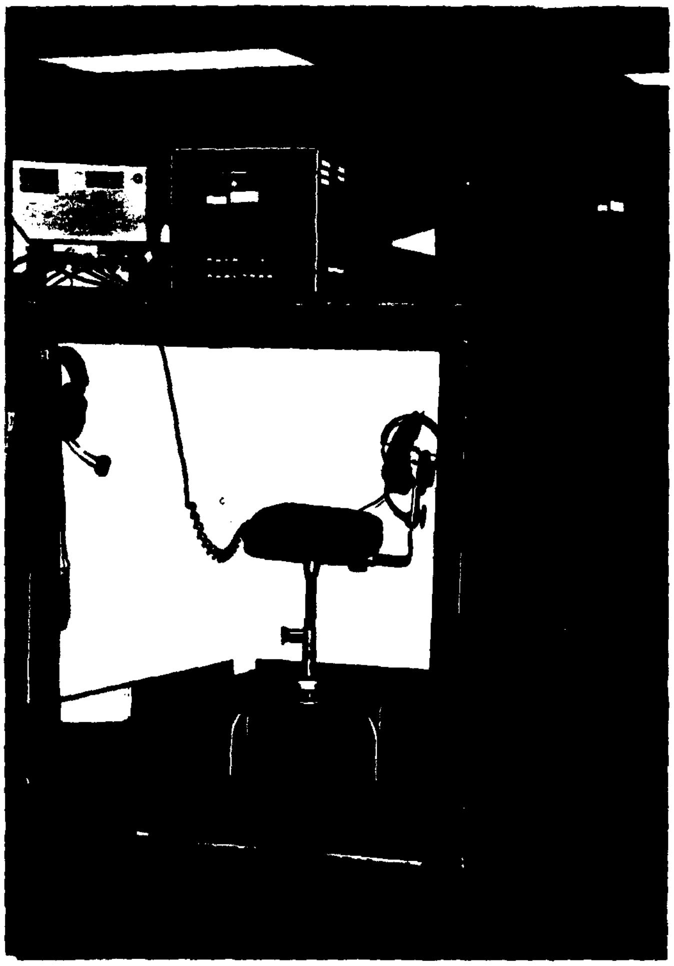
Polhemus. 1987. *3SPACE ISOTRAK User's Manual.* Colchester, VT. (May).

BIOGRAPHY

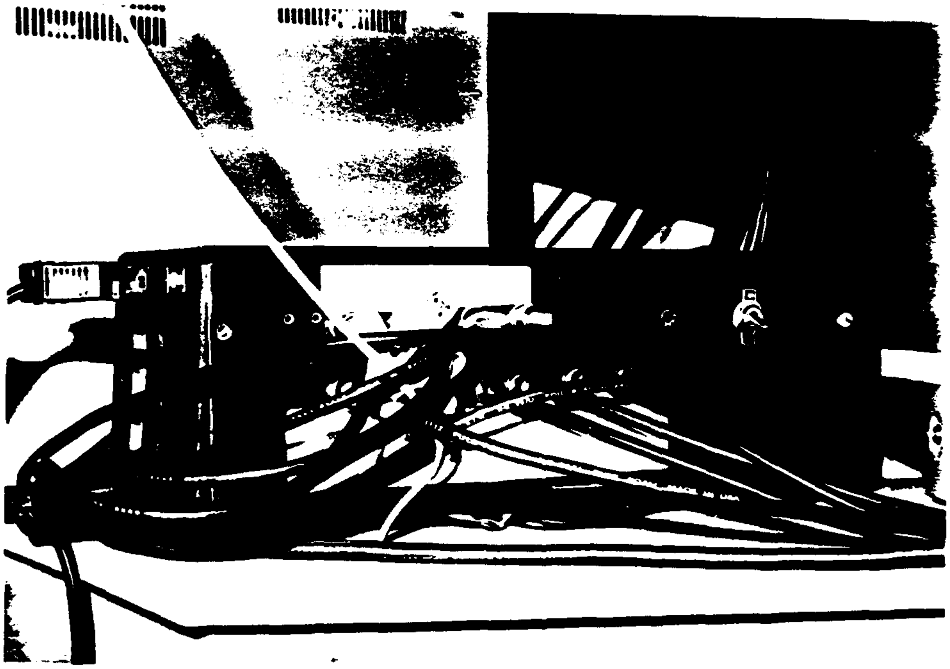
Mr. Dunn-Roberts is a Visual Systems Scientist at the Visual Systems Laboratory at the Institute for Simulation and Training. He is the Project Leader for the development and evaluation of advanced display technologies for use with real-time image generators. He is also involved in developing IST's capabilities in Virtual Environment research. Mr. Dunn-Roberts holds a Bachelors degree in Computer Science from the University of Central Florida, and is working on his Masters degree.

Appendix B:

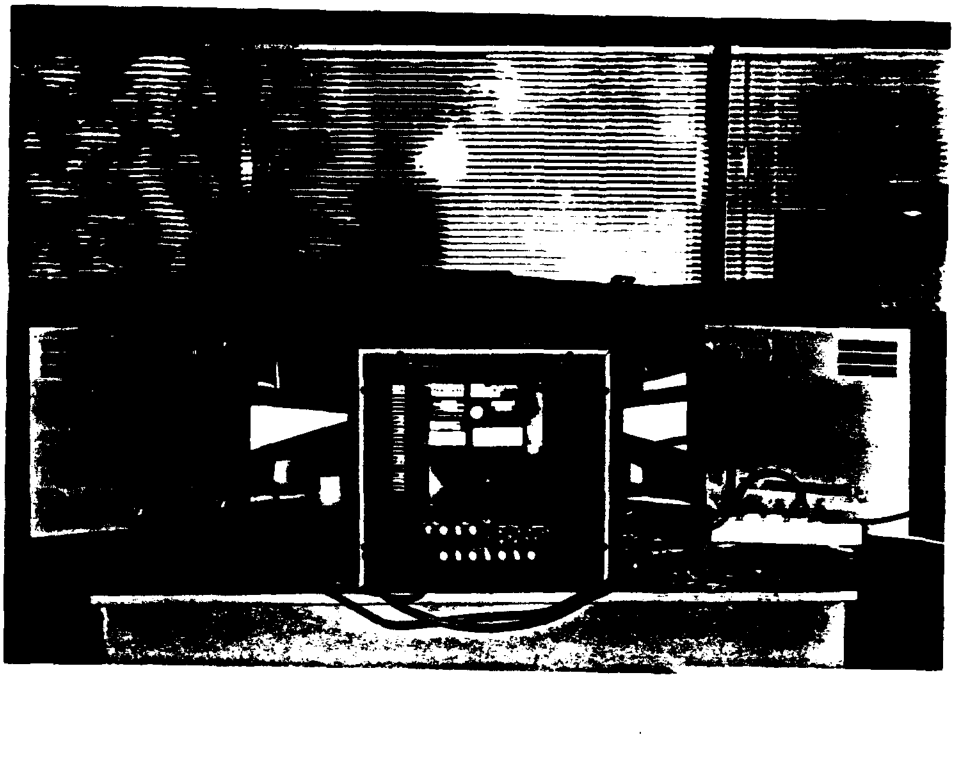
Photographs of Head Tracking Display



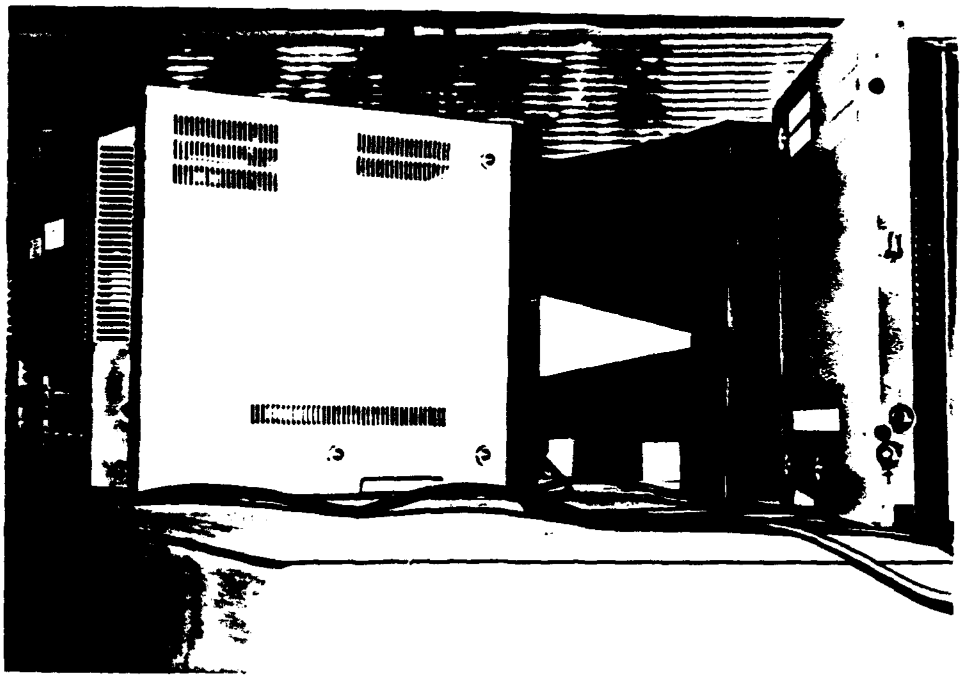
Head Tracking Display



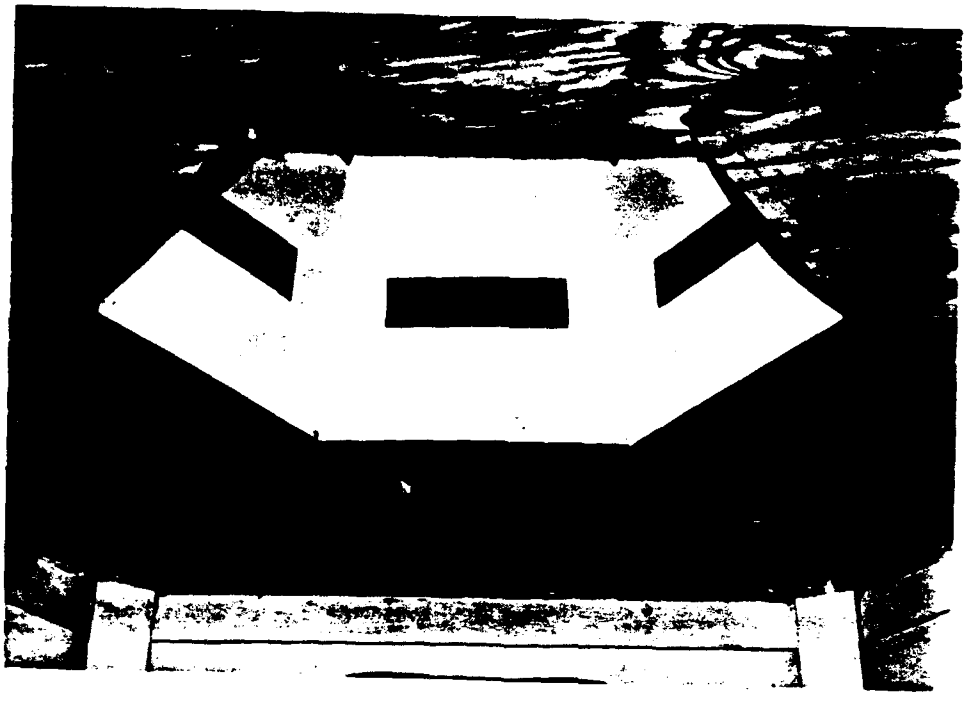
Video Switcher



Monitor Ring



Single Monitor and Viewport, Exterior View



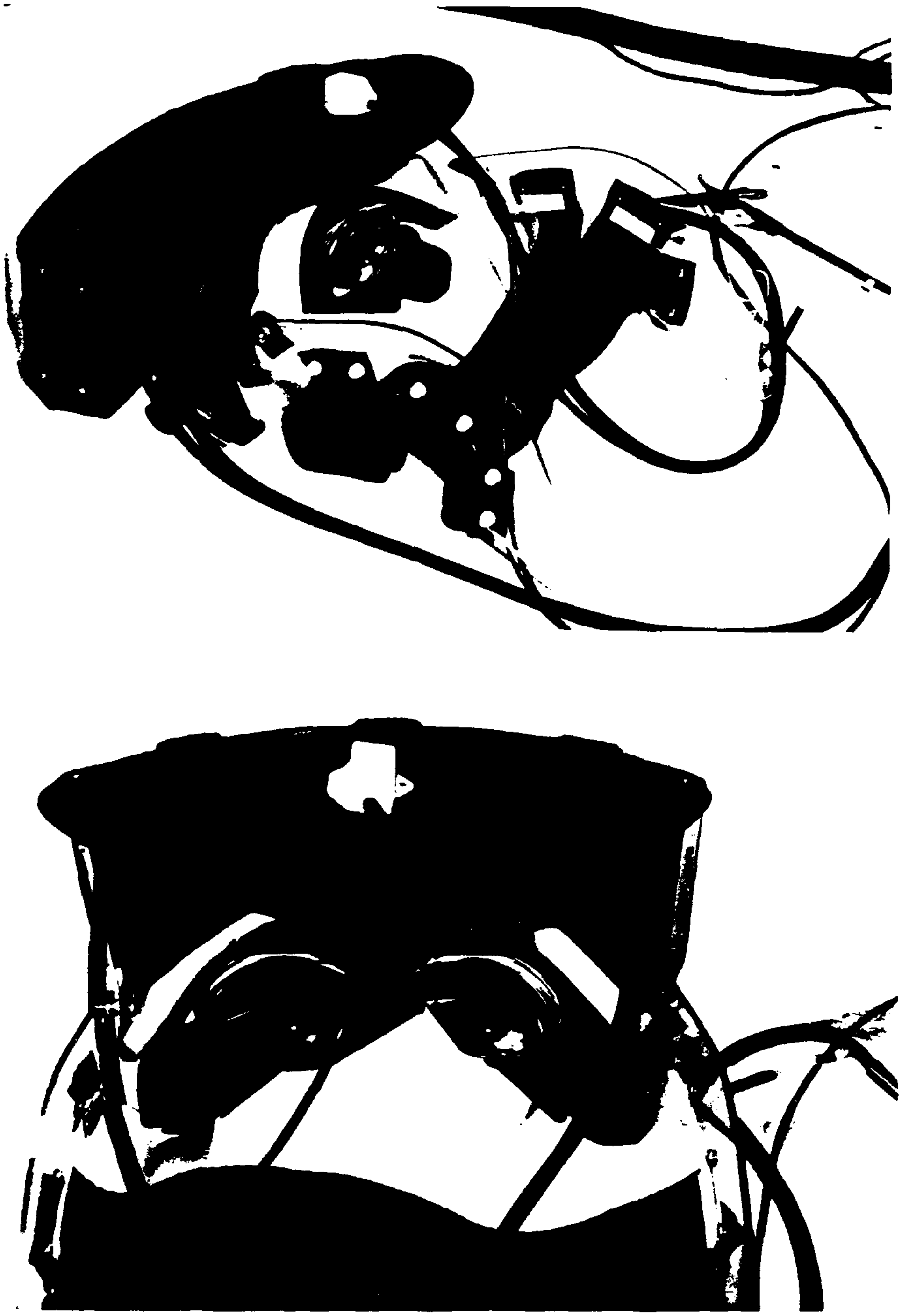
Viewports, Interior View

Subject and Experimenter

Appendix C:

Photographs of Head Mounted Displays

VPL Eyephones



Cyberface II

Appendix D:

Software to Control HTD Video Switcher

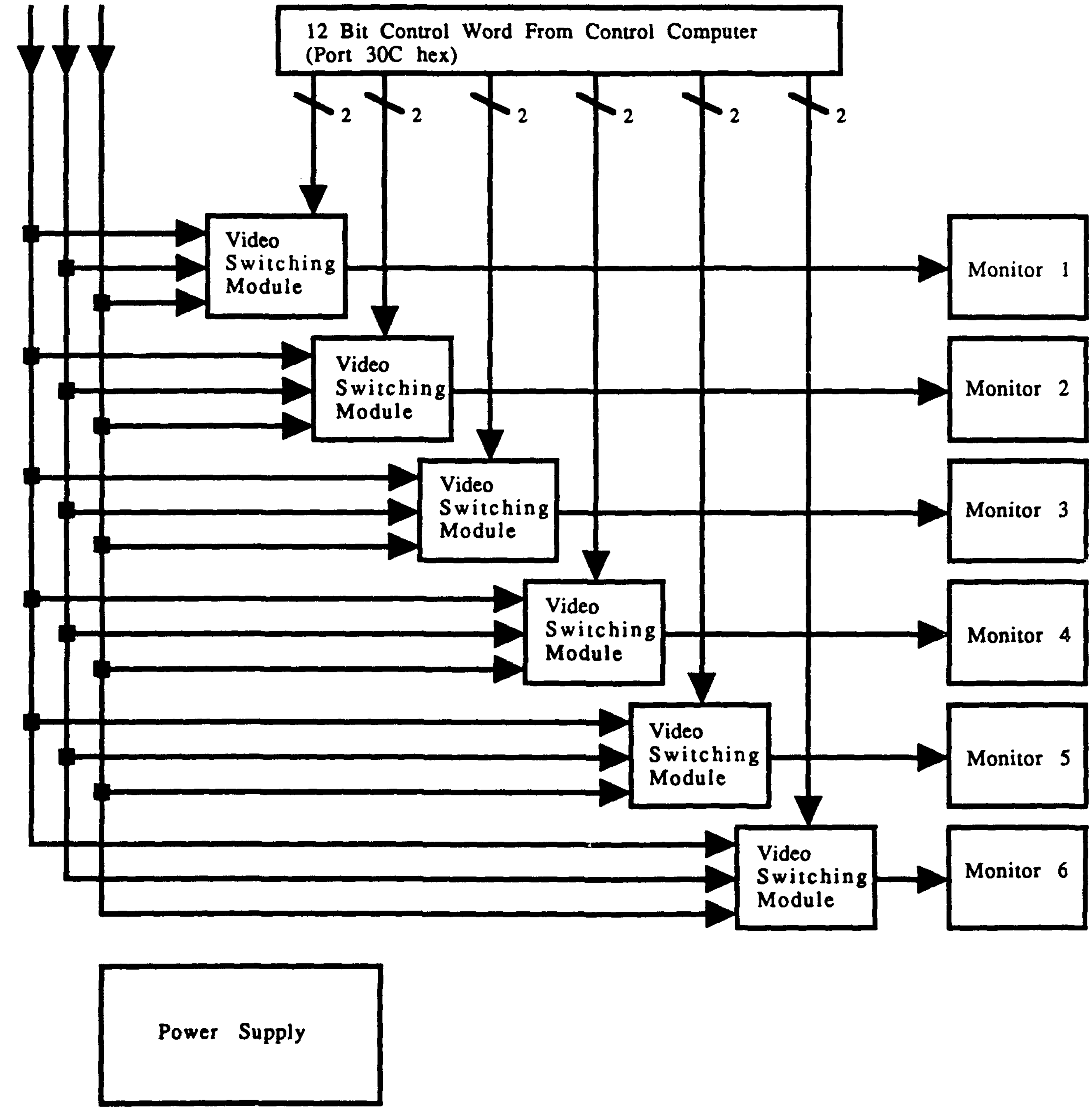
```
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/*                                                                      */
/*      HEAD TRACKING DISPLAY - Video Switcher Control Program          */
/*                                                                      */
/*                                                                      */
/*      FILENAME:  switcher.c                                           */
/*                                                                      */
/*                                                                      */
/*      By:     - Visual Systems Laboratory                             */
/*              - Institute for Simulation and Training                 */
/*              - University of Central Florida                         */
/*                                                                      */
/*                                                                      */
/*      Copyright (c) 1991 the University of Central Florida            */
/*              - All Rights Reserved                                   */
/*                                                                      */
/*                                                                      */
/*      Authors:        Marty Altman                                    */
/*                      Richard Dunn-Roberts                            */
/*                                                                      */
/*                                                                      */
/*      FUNCTION LIST:                                                  */
/*      -------------                                                   */
/*                                                                      */
/*      FUNC: void interrupt (*old_timer_routine)(void);                */
/*              a pointer to the old timer routine                      */
/*      FUNC: void interrupt handle_timer_interrupt(void);              */
/*              our new timer interrupt handler                         */
/*      FUNC: void set_timer(void);                                     */
/*              to reprogram the clock timer to interrupt at 60Hz       */
/*              instead of 18.2Hz                                       */
/*      FUNC: void reset_timer(void);                                   */
/*              to reprogram the clock timer back to 18.2Hz             */
/*      FUNC: void readConfig(void);                                    */
/*              reads configuration information from external file      */
/*      FUNC: void initPolhemus(void);                                  */
/*              initializes the Polhemus                                */
/*      FUNC: void getRecord(void);                                     */
/*              gets a data record from the Polhemus                    */
/*                                                                      */
/*                                                                      */
/*      General Comments:                                               */
/*              This program was written to run on a PC-AT, using       */
/*              Borland C++ version 2.0 (with the built-in assembler).  */
/*              It is designed to control the operation of the Head     */
/*              Tracking Display system.                                */
/*                                                                      */
/*                                                                      */
/*      Operational Comments:                                           */
/*              The switcher control program reads the head orientation */
/*              from the Polhemus magnetic sensor and the video sync    */
/*              signal from the SIMNET Image Generator.  It then        */
/*              determines which screens should be turned on (sending a */
/*              control word to the Video Switcher), and determines the */
/*              appropriate voltage to emulate the cupola's             */
/*              potentiometer (sending a control word to a digital-     */
/*              to-analog converter, and the analog signal is then      */
/*              routed to the SIMNET controls).  See the figure below.  */
/*                                                                      */
/*                                                                      */
/*              The basic layout is as follows:                         */
/*                                                                      */
/*      +-----------------+    15Hz sync signal +-----------------+     */
/*      |      PC_AT      |<--------------------- SIMNET MASSCOMP |     */
/*      |                 ------------------+   |                 |     */
/*      |   {switcher}    --------------+   |   |                 |     */
/*      |   {program }    |<------+     |   |   +-----------------+     */
/*      |                 |<--+   |     |   |                           */
/*      +-----------------+   |   |     |   |                           */
/*                            |   |     |   |                           */
/*                            |   |     |   |                           */
/*           head orientation |   |     |   | which screens to          */
/*                            |   |     |   | turn on                   */
/*      +-----------------+   |   |     |   |   +-----------------+     */
/*      |    POLHEMUS     |   |   |     |   |   | VIDEO SWITCHER  |     */
/*      |                 ----+   |     |   +-->|                 |     */
/*      |                 |       |     |       |                 |     */
/*      +-----------------+       |     |       +-----------------+     */
/*                                |     |                               */
/*                                |     | emulation of potentiometer    */
/*              video sync signal |     | (analog voltage)              */
/*                                |     |                               */
/*      +-----------------+       |     |       +-----------------+     */
/*      |    SIMNET IG    |       |     |       | SIMNET CONTROLS |     */
/*      |                 --------+     +------>|                 |     */
/*      |                 |                     |                 |     */
/*      +-----------------+                     +-----------------+     */
/*                                                                      */
/*                                                                      */
/*      For further system details, refer to the project report.        */
/*                                                                      */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/


/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/
/* Type, Structure & Constant Defs */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/

#define BLANK_SWITCHER_WORD 0FFFh


/*=-=-=-=-=-=-=-=-=-=-=-=-=*/
/* Necessary Include Files */
/*=-=-=-=-=-=-=-=-=-=-=-=-=*/

#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include "comport.h"
#include "polhemus.h"


/*=-=-=-=-=-=-=-=-=-=-=*/
/* Function Prototypes */
/*=-=-=-=-=-=-=-=-=-=-=*/

void set_timer(void);
void reset_timer(void);
void interrupt handle_timer_interrupt(void);
void readConfig(void);
void initPolhemus(void);
void getRecord(void);


/*=-=-=-=-=*/
/* Globals */
/*=-=-=-=-=*/

void interrupt (*old_timer_routine)(void);
int     switcher_control_word;
int     switcher_control[6] =
                { 0x0951,0x0546,0x0519,0x0465,0x0195,0x0654 };
int     TC_yaw = 0;
int     center_screen = 0;
int     last_center_screen = 1;
int     cig_angle_control_word;
int     cig_angle_control[6] =
                { 0x07FF,0x0AA9,0x0D54,0x0000,0x02AA,0x0554 };
int     screen[360] =
                {3,3,3,3,3,3,3,3,3,3, 3,3,3,3,3,3,3,3,3,3, 3,3,3,3,3,3,3,3,3,3,
                 2,2,2,2,2,2,2,2,2,2, 2,2,2,2,2,2,2,2,2,2, 2,2,2,2,2,2,2,2,2,2,
                 2,2,2,2,2,2,2,2,2,2, 2,2,2,2,2,2,2,2,2,2, 2,2,2,2,2,2,2,2,2,2,
                 1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,1,
                 1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,1,
                 0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,
                 0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,
                 5,5,5,5,5,5,5,5,5,5, 5,5,5,5,5,5,5,5,5,5, 5,5,5,5,5,5,5,5,5,5,
                 5,5,5,5,5,5,5,5,5,5, 5,5,5,5,5,5,5,5,5,5, 5,5,5,5,5,5,5,5,5,5,
                 4,4,4,4,4,4,4,4,4,4, 4,4,4,4,4,4,4,4,4,4, 4,4,4,4,4,4,4,4,4,4,
                 4,4,4,4,4,4,4,4,4,4, 4,4,4,4,4,4,4,4,4,4, 4,4,4,4,4,4,4,4,4,4,
                 3,3,3,3,3,3,3,3,3,3, 3,3,3,3,3,3,3,3,3,3, 3,3,3,3,3,3,3,3,3,3};
int       comPort;
int       baudRate;
char      initString[100];
char      termString[100];
int       high = 0;
int       sync = 0;
int       screen_changed = 0;
int       change_the_switcher = 0;
int       delay_cycles = 10;
int       delay_counter = 0;


/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/
/* Function set_timer                                                       */
/*                                                                          */
/* PARAMETERS:                                                              */
/* void                                                                     */
/*                                                                          */
/* PROCESS:                                                                 */
/*      This routine is used to reprogram the clock timer to interrupt      */
/*      at 60Hz.  Note that it is virtually impossible to accurately        */
/*      detect a 15Hz signal when checking every 18.2Hz (the normal         */
/*      setting for the clock timer interrupt on a PC).  Also note          */
/*      that because the PC for this project was dedicated, it was          */
/*      not a problem for that PC's sense of time to be distorted.          */
/*                                                                          */
/* RETURN VALUE:                                                            */
/* void                                                                     */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/

void set_timer(void) {        /* program timer to interrupt at 60 Hz */
        asm {
                cli
                push    ax
                mov     al,00110110b
                out     43h,al

                mov     ax,19886
                out     40h,al
                mov     al,ah
                out     40h,al
                pop     ax
                sti
        }
```

```
}


/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* Function reset_timer                                                 */
/*                                                                      */
/* PARAMETERS:                                                          */
/* void                                                                 */
/*                                                                      */
/* PROCESS:                                                             */
/*      This routine is used to reprogram the clock timer to interrupt  */
/*      at 18.2Hz (the normal setting for a PC).                        */
/*                                                                      */
/* RETURN VALUE:                                                        */
/* void                                                                 */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/

void reset_timer(void) {     /* program timer to interrupt at 18.2 Hz */
        asm {
                cli
                push   ax
                mov    al,00110110b
                out    43h,al

                mov    ax,0
                out    40h,al
                mov    al,ah
                out    40h,al
                pop    ax
                sti
        }
}


/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* Function handle_timer_interrupt                                      */
/*                                                                      */
/* PARAMETERS:                                                          */
/* void                                                                 */
/*                                                                      */
/* PROCESS:                                                             */
/*      This interrupt routine is installed on the clock timer (0x1C),  */
/*      and is used to monitor the joystick port.  The joystick port    */
/*      is where the 15Hz sync signal from the SIMNET MASSCOMP is       */
/*      brought in.  The routine looks for the falling edge of the      */
/*      signal, and when detected sets a flag called sync to 1.         */
/*      Note that in order for this scheme to work, the clock timer     */
/*      must have been 'sped up'.                                       */
/*                                                                      */
/* RETURN VALUE:                                                        */
/* void                                                                 */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/

void interrupt handle_timer_interrupt(void) {
        asm {
                mov    dx,201h
                in     al,dx
                test   al,20h
                jz     gone_low
                mov    high,1
        }
        return;

gone_low:
        if (high) {
                asm {
                        mov    sync,1
                        mov    high,0
                }
        }
        return;
}


/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* Function main                                                      */
/*                                                                    */
/* PARAMETERS:                                                        */
/* void                                                               */
/*                                                                    */
/* PROCESS:                                                           */
/*      This is the main routine.                                     */
/*                                                                    */
/* RETURN VALUE:                                                      */
/* void                                                               */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/

void main(void) {
        /**** banner ****/
        clrscr();
        printf("Head Tracking Display Control Program.\n");
        printf("\n\nPress any key to exit.");

        /**** set initial values ****/
        switcher_control_word  = switcher_control[0];
        cig_angle_control_word = cig_angle_control[0];

        /**** other initialization ****/
        readConfig();
        initPolhemus();

        /**** take over the timer interrupt ****/
        old_timer_routine = getvect(0x1C);
        setvect(0x1C, handle_timer_interrupt);

        /**** reprogram to 60Hz ****/
        set_timer();

        /******** begin MAIN LOOP ********/
        while (!kbhit()) {

                /**** get a record from the Polhemus ****/
                getRecord();

                /**** determine center screen ****/
                center_screen = screen[TC_yaw];
                if (center_screen != last_center_screen) {
                        switcher_control_word  = switcher_control[center_screen];
                        cig_angle_control_word = cig_angle_control[center_screen];
                        asm {
                                mov    ax,center_screen
                                mov    last_center_screen,ax
                                mov    screen_changed,1
                        }
                }

                /**** if we need to update the cig angle ****/
                if (sync) {
                        if (screen_changed) {
                                asm {
                                        mov     ax,BLANK_SWITCHER_WORD
                                        mov     dx,030Ch
                                        out     dx,ax
                                        mov     ax,delay_cycles
                                        mov     delay_counter,ax
                                        mov     screen_changed,0
                                        mov     change_the_switcher,1
                                }
                        }
                        asm {
                                mov     ax,cig_angle_control_word
                                mov     dx,0302h
                                out     dx,ax
                                mov     sync,0
                        }
                }

                /**** if we need to update the video switcher ****/
                if (change_the_switcher) {
                        if (delay_counter) {
                                asm {
                                        dec     delay_counter
                                }
                        }
                        else {
                                asm {
                                        mov     ax,switcher_control_word
                                        mov     dx,030Ch
                                        out     dx,ax
                                        mov     change_the_switcher,0
                                }
                        }
                }

        }   /******** end MAIN LOOP ********/

        /**** clean up after ourselves ****/
        reset_timer();
        setvect(0x1C, old_timer_routine);
        DeinstallDrivers();
}


/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* Function getRecord                                                   */
/*                                                                      */
/* PARAMETERS:                                                          */
/* void                                                                 */
/*                                                                      */
/* PROCESS:                                                             */
/*      This routine is used to get a data record from the Polhemus.    */
/*      Since we are only interested in the orientation about the       */
/*      vertical axis, the only value we update is the TC_yaw.          */
/*                                                                      */
/* RETURN VALUE:                                                        */
/* void                                                                 */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/

void getRecord(void) {
        int     retcode, j, temphex, tempint;
        char    data[255], input[255], temp, recordType;
        float   yaw;
        ReceiveData( comPort, data, 18 );
        if ( !( data[0] & 0x80 ) ) {
                while ( !( data[0] & 0x80 ) )
                        ReceiveData( comPort, data, 1 );
                ReceiveData( comPort, (data+1), 17 );
        }

        for ( j = 0; j < 7; j++ ) {
                data[j] = (data[j] & 0x7F) | ( ( data[7] & 0x01 ) << 7 );
                data[7] = data[7] >> 1;
        }

        for ( j = 8; j < 15; j++ ) {
                data[j] = (data[j] & 0x7F) | ( ( data[15] & 0x01 ) << 7 );
                data[15] = data[15] >> 1;
        }

        data[16] = (data[16] & 0x7F) | ( ( data[17] & 0x01 ) << 7 );
        data[15] = data[16];

        tempint = *(int *)(data+10);
        yaw = (((float)tempint)*230.0/32767.0)+180.0;
        yaw = yaw < 0.0 ? 0.0 : yaw;
        yaw = yaw > 360.0 ? 360.0 : yaw;
        TC_yaw = ( RABS( TC_yaw - yaw ) ) < EPSILON ? TC_yaw : yaw;
}


/*----------------------------------------------------------------------*/
/* Function readConfig                                                  */
/*                                                                      */
/* PARAMETERS:                                                          */
/* void                                                                 */
/*                                                                      */
/* PROCESS:                                                             */
/*      This routine is used to read values from an external config     */
/*      file.                                                           */
/*                                                                      */
/* RETURN VALUE:                                                        */
/* void                                                                 */
/*----------------------------------------------------------------------*/

void readConfig(void) {
        FILE    *configFile;
        char    configBuffer[129], *configType, *configValue;
        int     i;

        if((configFile = fopen("config.dat","rt")) == NULL) {
                fprintf(stderr, "Configuration file not found\n");
                exit(-1);
        }

        while (fgets(configBuffer,128,configFile) != NULL) {

                if ((configType = strtok(configBuffer," \n\t")) != NULL) {
                        if ((configValue = strtok(NULL," \n\t")) == NULL) {
                                printf("value not found for type = %s\n",
                                        configType);
                                exit(-1);
                        }
                }
                else continue;

                if ( !strcmp( configType, "polhemusComPort" ) )
                        comPort = atoi( configValue );
                if ( !strcmp( configType, "polhemusBaudRate" ) )
                        baudRate = atoi( configValue );

                if ( !strcmp( configType, "delay_cycles" ) ) {
                        delay_cycles = atoi( configValue );
                        if (delay_cycles==0) delay_cycles=10;
                        delay_counter = delay_cycles;
                }

                if ( !strcmp( configType, "polhemusInitString" ) )
                        strcpy( initString, configValue );

                if ( !strcmp( configType, "controlWords" ) ) {
                        sscanf( configValue, "%x", &switcher_control[0] );
                        switcher_control_word = switcher_control[0];
                        for (i=1; i<6; i++) {
                                if ((configValue=strtok(NULL," \n\t"))==NULL) {
                                        printf("value not found for type = %s\n",
                                                configType);
                                        exit(-1);
                                }
                                sscanf(configValue,"%x",&switcher_control[i]);
                        }
                }
        }

        fclose(configFile);
}


/*----------------------------------------------------------------------*/
/* Function initPolhemus                                                */
/*                                                                      */
/* PARAMETERS:                                                          */
/* void                                                                 */
/*                                                                      */
/* PROCESS:                                                             */
/*      This routine is used to initialize the Polhemus.                */
/*                                                                      */
/* RETURN VALUE:                                                        */
/* void                                                                 */
/*----------------------------------------------------------------------*/

void initPolhemus(void) {
        char    *commandPtr, command[30];
        int     commandLength;

        InitComPort( comPort, DIVISOR( baudRate ) );
        LowerDTR( comPort );
        commandPtr = strtok( initString, ";" );

        while(commandPtr) {
                strcpy( command, commandPtr );

                if ( ( commandLength = strlen( command ) ) != 1 ) {
                        command[commandLength] = 13;
                        command[++commandLength] = 0;
                }

                TransmitData(  comPort, command, commandLength );
                commandPtr = strtok( NULL, ";" );
        }

        RaiseDTR(comPort);
```

Appendix E:

Schematics of HTD Video Switcher

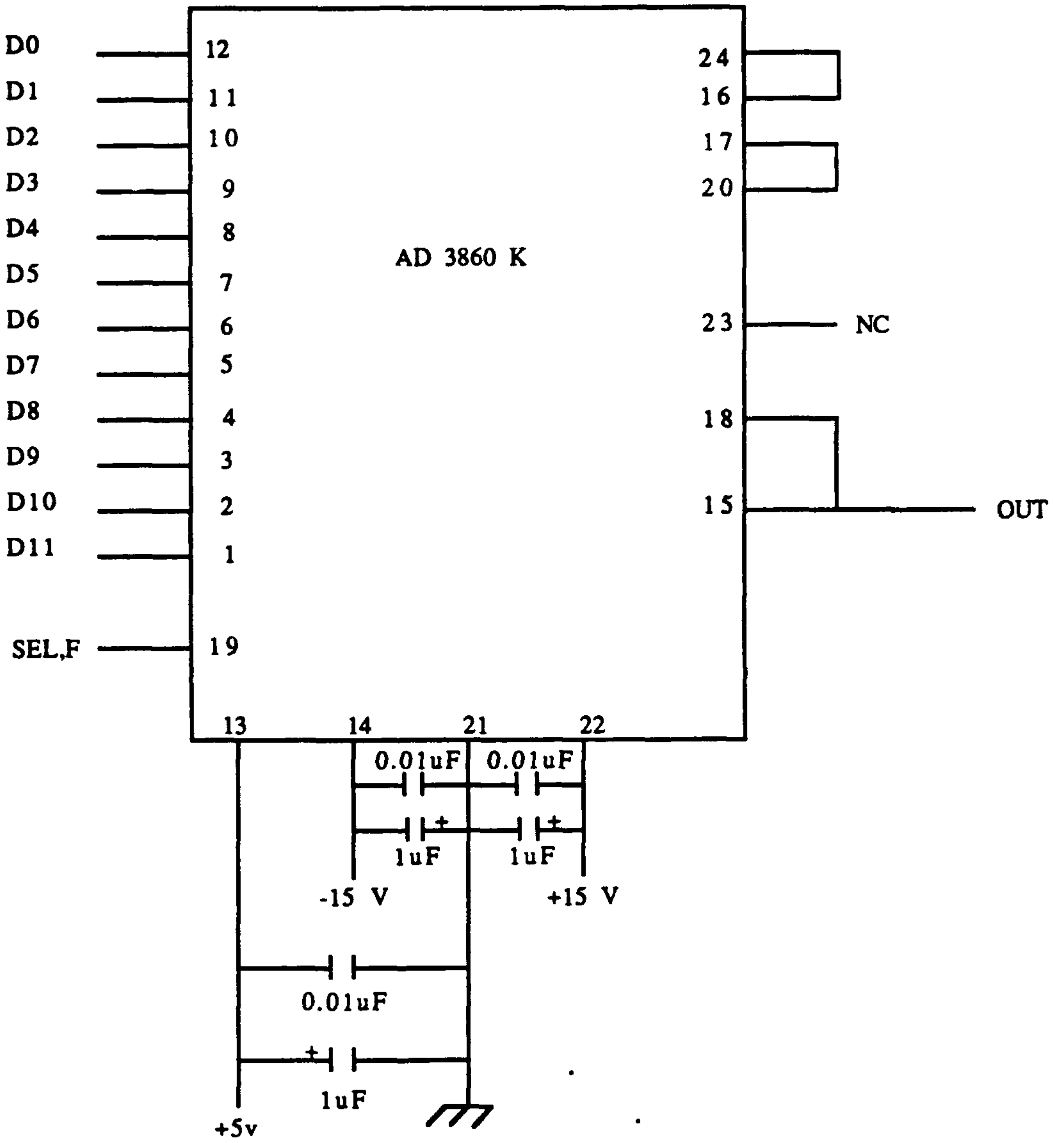
Video Switcher Block Diagram



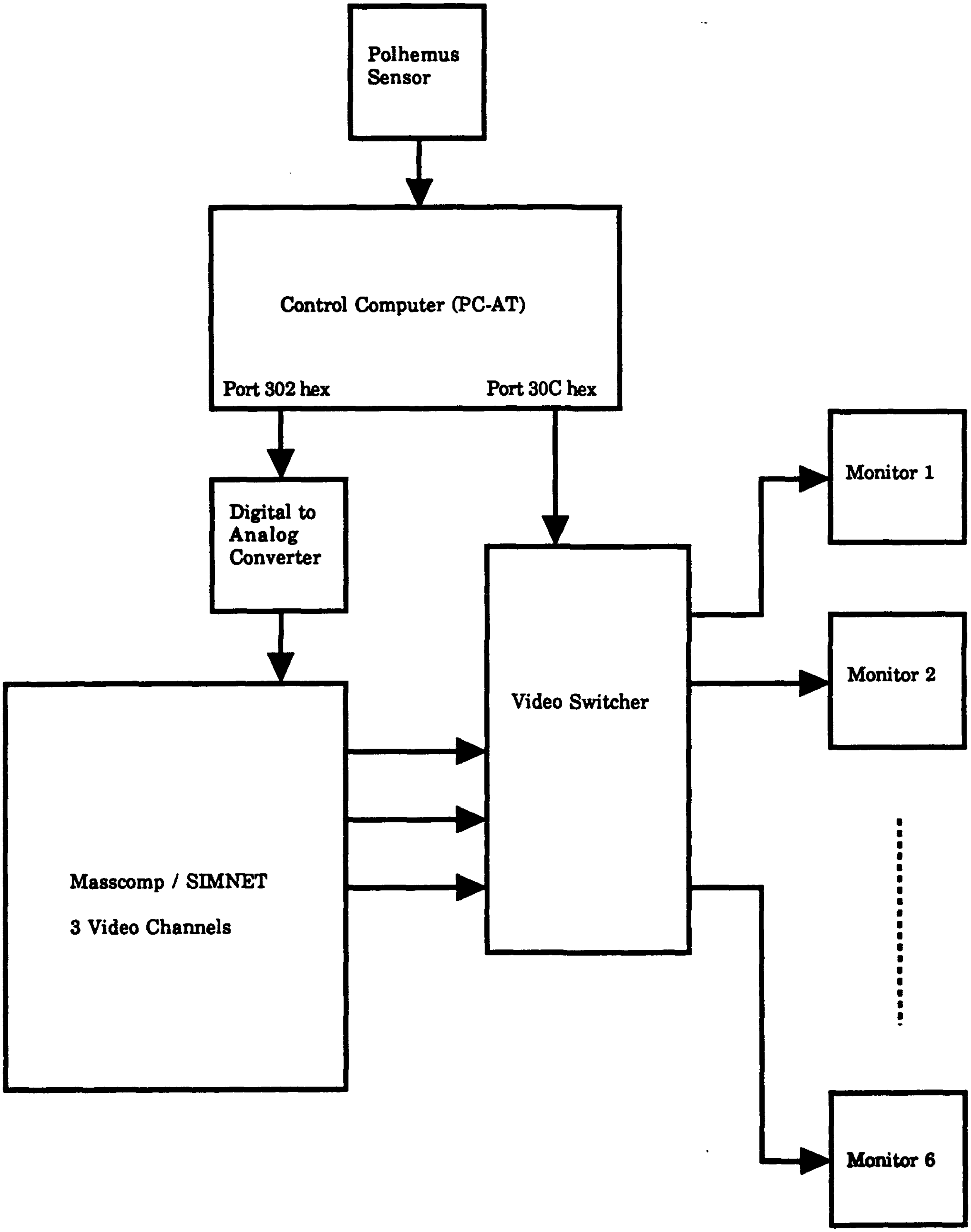
NOTES: 1) Any monitor can be switched to any video channel, or remain blank.

2) Circuitry prevents attempts to switch two channels to same monitor.

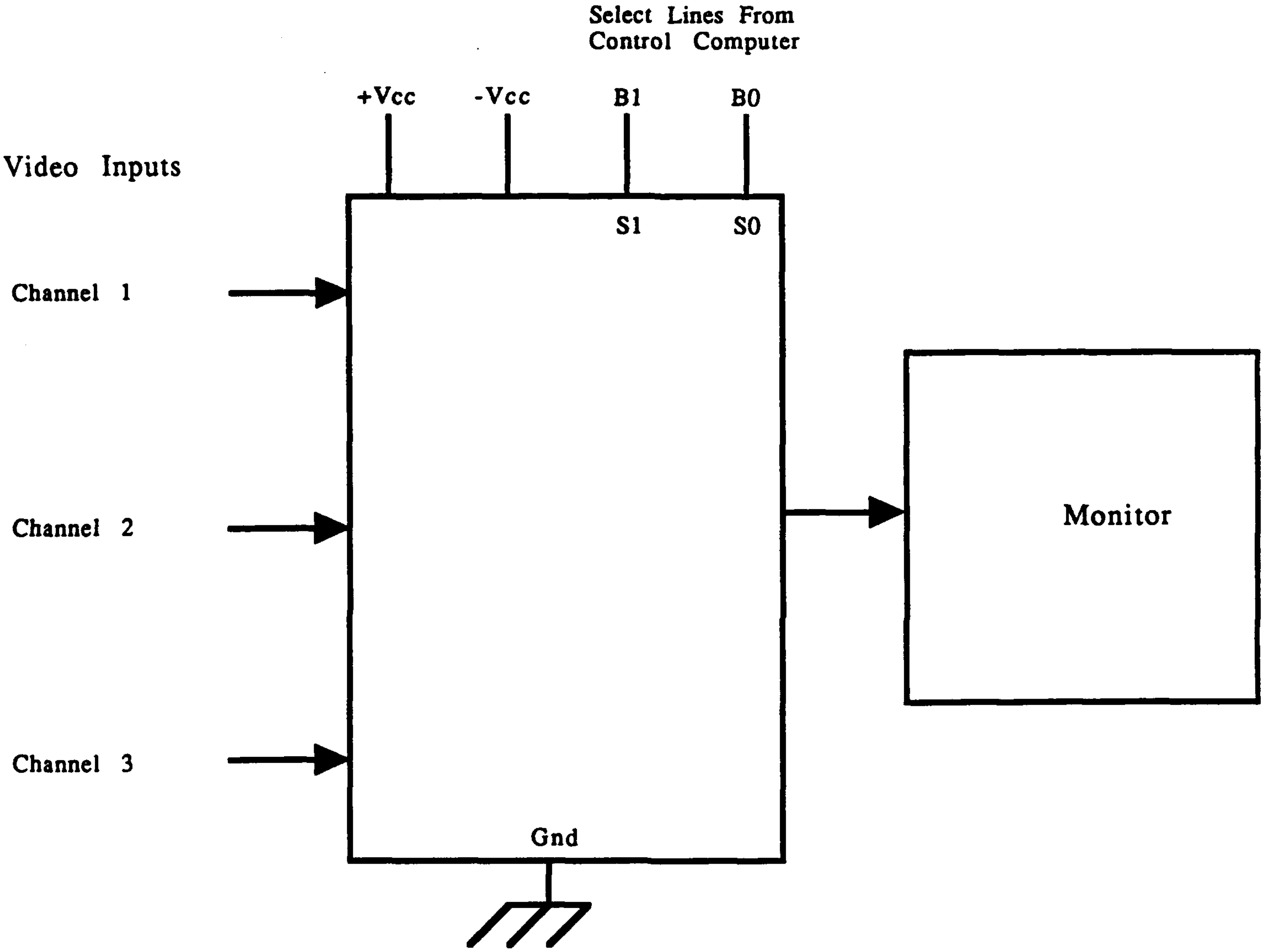
D/A CONVERTER



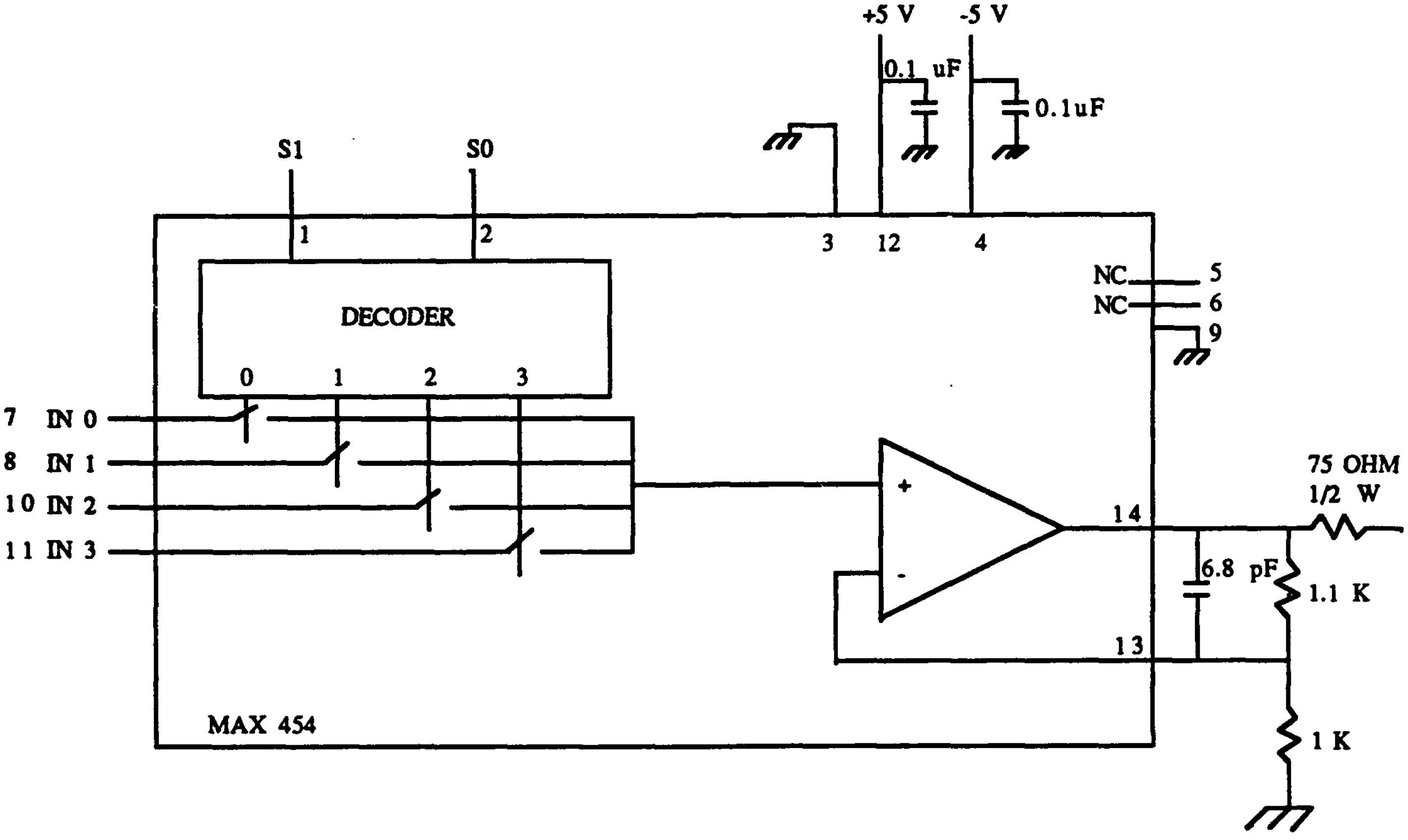
HTD
Block Diagram



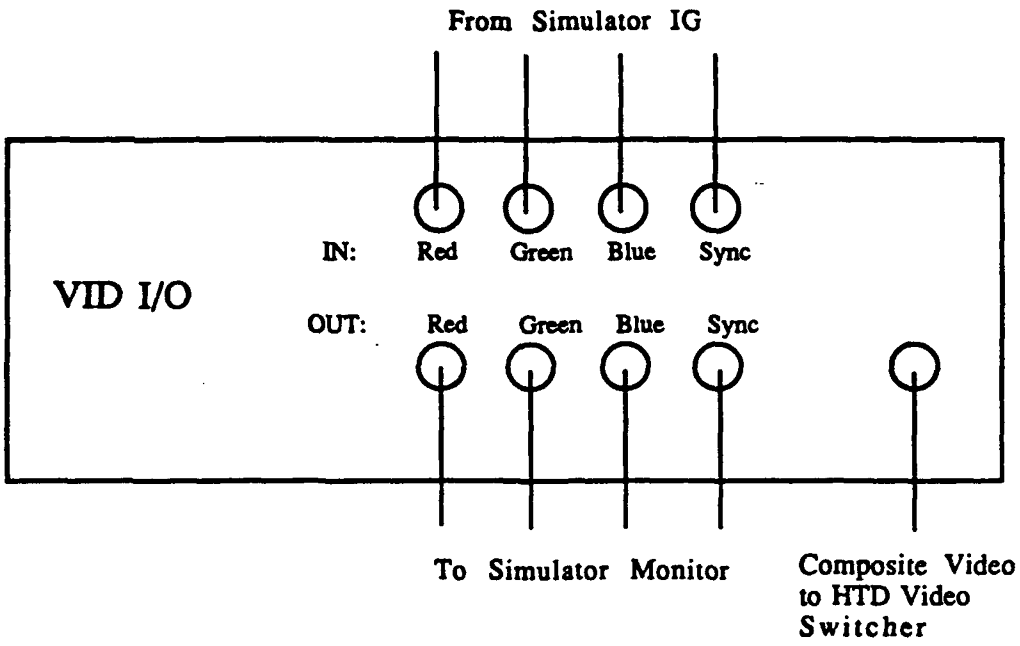
Video Switching Module Block Diagram (1 of 6)

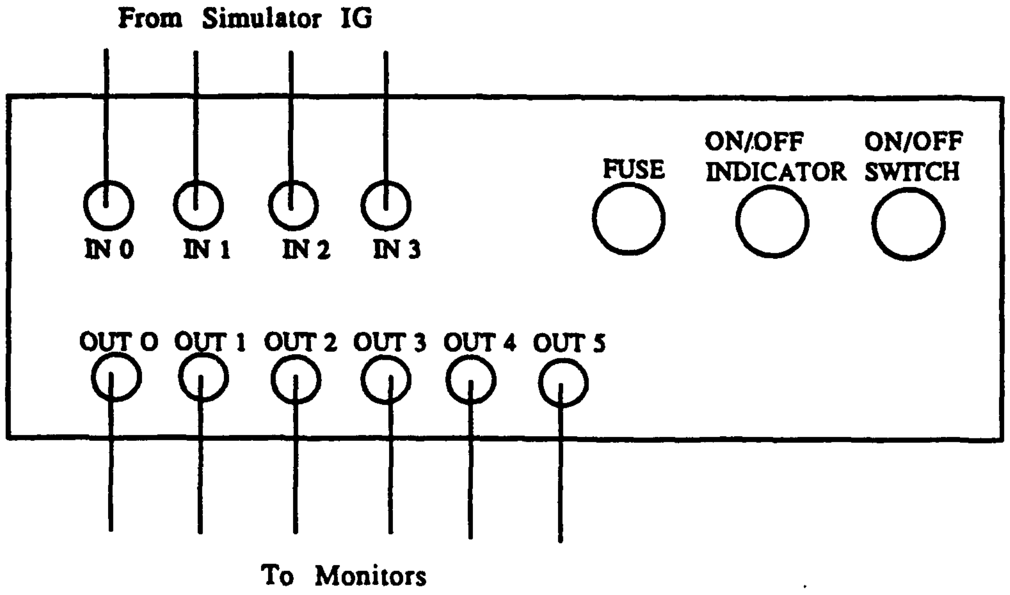


MAX 454 VIDEO MULTIPLEXER/AMPLIFIER

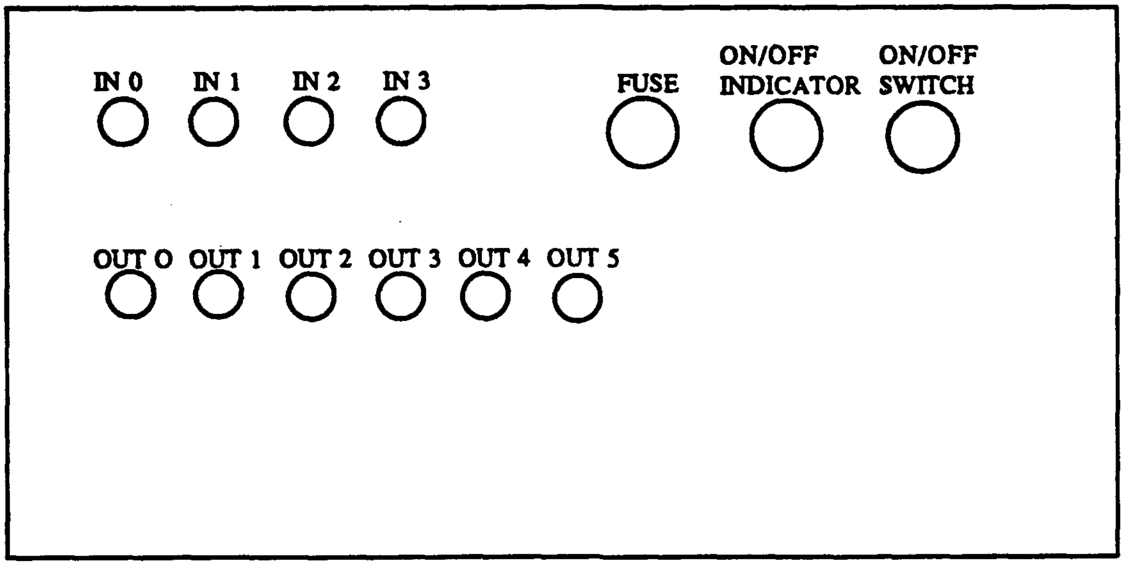


Video switcher: one module
Functional diagram

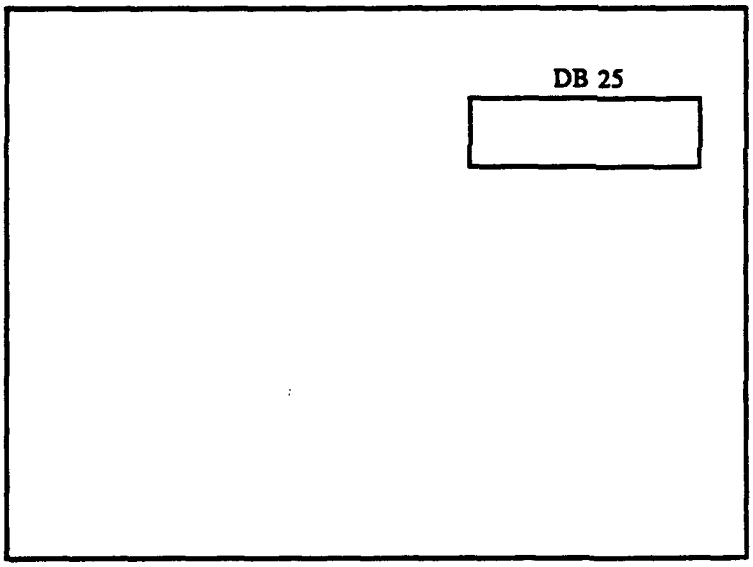

From Simulator IG
VID I/O
IN:
Red
Green
Blue
Sync
OUT:
Red
Green
Blue
Sync
To Simulator Monitor
Composite Video to HTD Video Switcher


From Simulator IG
FUSE
ON/OFF INDICATOR
ON/OFF SWITCH
IN 0
IN 1
IN 2
IN 3
OUT 0
OUT 1
OUT 2
OUT 3
OUT 4
OUT 5
To Monitors

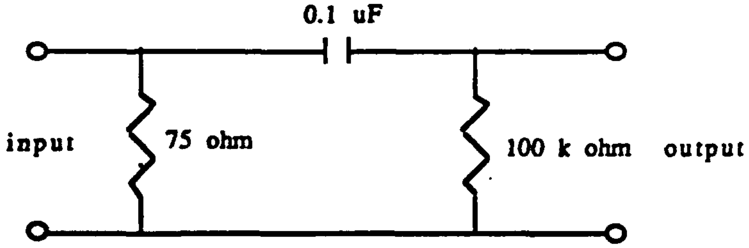
VIDEO SWITCHER; SIDE VIEW



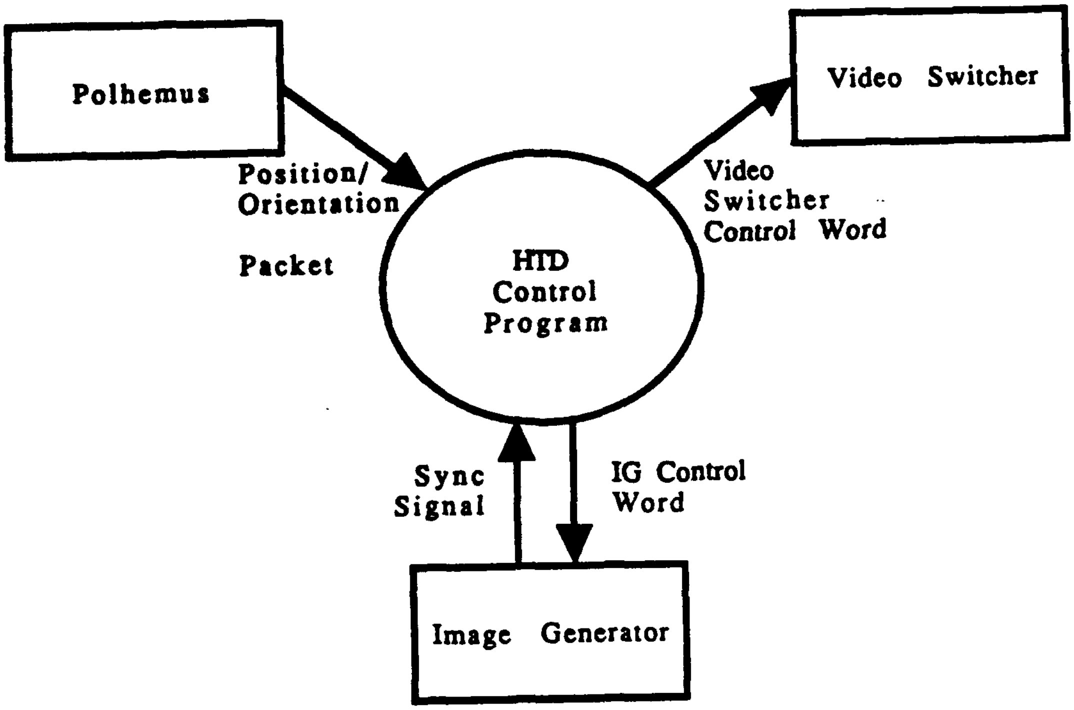
VIDEO SWITCHER; FRONT VIEW



Input circuit for video switcher

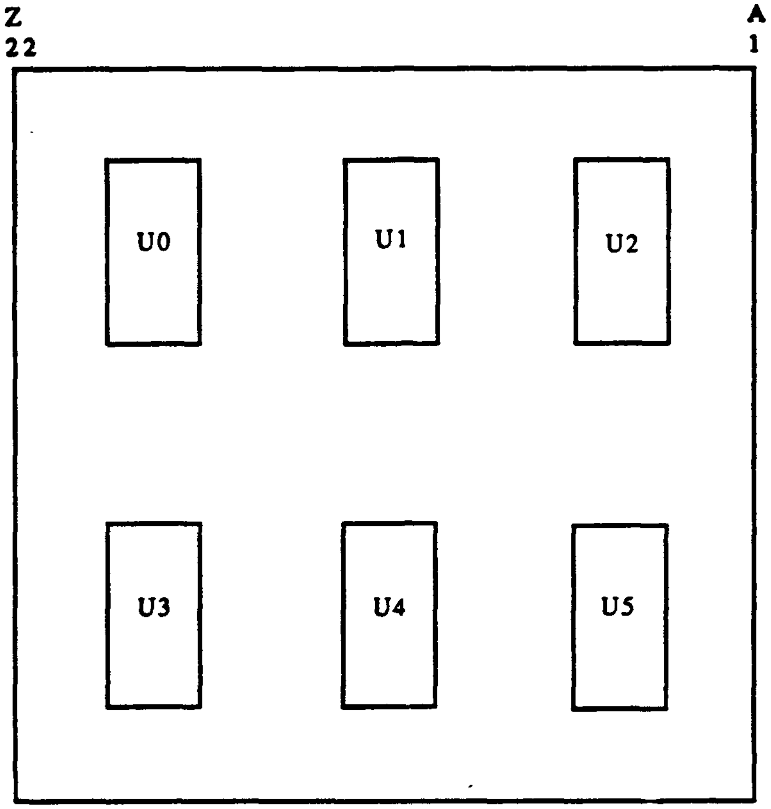


Note: The 100k resistor and the 0.1uF capacitor were added in order to eliminate a D.C. voltage that was added to the video signal from the Vid I/O boxes.


Polhemus
Position/ Orientation
Packet
Video Switcher
Video Switcher Control Word
HTD Control Program
Sync Signal
IG Control Word
Image Generator

The following diagram shows the physical layout of the MAX 454 video switching chips.

CB 1



Note: see circuit diagram of MAX 454 for connections of individual video switchers U0-U5.

Appendix F:

Software to Control SIMNET Head Tracking Display

```
/*--------------------------------------------------------------*/
/*              SIMNET M1 with integrated Polhemus              */
/*                                                              */
/* FILENAME:  m1_polhemus.c                                     */
/*                                                              */
/*                                                              */
/* By   - MaK Technologies                                      */
/*        and                                                   */
/*      - Visual Systems Laboratory                             */
/*      - Institute for Simulation and Training                 */
/*      - University of Central Florida                         */
/*                                                              */
/*                                                              */
/* Copyright (c) 1991 MaK Technologies and                      */
/*              the University of Central Florida               */
/*      - All Rights Reserved                                   */
/*                                                              */
/*                                                              */
/* Authors:  Warren Katz and Marty Altman                       */
/*                                                              */
/*                                                              */
/* PUBLIC FUNCTION LIST:                                        */
/* FUNC:  void polhemus_init() - initialize Polhemus            */
/* FUNC:  void polhemus_simul() - read and stuff a value for    */
/*              the z rotation                                  */
/* FUNC:  void polhemus_exit() - shutdown the Polhemus          */
/*                                                              */
/*                                                              */
/* PRIVATE FUNCTION LIST:                                       */
/* FUNC:  int open_polhemus(char *device_name) - actual open    */
/* FUNC:  int close_polhemus() - reset port                     */
/* FUNC:  int copy_termio(struct termio source, dest) - copy    */
/*                                                              */
/*                                                              */
/* General Comments:                                            */
/*      This file integrates the Polhemus Head Tracker with     */
/*      the M1 cupola, bypassing the DTAD card.  The Polhemus   */
/*      head tracker computer sends a Z rotation over an        */
/*      RS-232 interface.                                       */
/*                                                              */
/*--------------------------------------------------------------*/

/*-----------------------------------*/
/* Type, Structure and Constant Defs */
/*-----------------------------------*/

#define FORMAT_ASCII            0
#define FORMAT_BINARY           1

#define UNIT_INCHES             0
#define UNIT_CENTIMETERS        1

#define DATA_PACKET_XYZ         0
#define DATA_PACKET_AER         1
#define DATA_PACKET_XYZAER      2

#define PACKET_SIZE     24


/*-------------------------*/
/* Necessary Include Files */
/*-------------------------*/

#include "stdio.h"
#include "fcntl.h"
```

```
#include "sim_dfns.h"
#include "sim_types.h"
#include "sim_macros.h"

#include <stdio.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/file.h>
#include <termio.h>
#include <unistd.h>
#include <fcntl.h>


/*---------*/
/* Globals */
/*---------*/

static float    azimuth = 0.0;
int             device, result, packet_size;
struct termio   polhemus_control, old_port_control;
char            temp_buffer[80];


/*------------------*/
/* PUBLIC functions */
/*------------------*/

/*--------------------------------------------------------------*/
/* Function polhemus_init                                       */
/*                                                              */
/* PARAMETERS:                                                  */
/*      void                                                    */
/*                                                              */
/* PROCESS:                                                     */
/*      This function is called to initialize the Polhemus on   */
/*      'tty2'.  If successful, a request is made for the first */
/*      data record.  (Writing a "P" is a request for data.)    */
/*                                                              */
/* RETURNS:                                                     */
/*      void                                                    */
/*--------------------------------------------------------------*/

void polhemus_init()
{
    if (open_polhemus("/dev/tty2") == 0) {
        printf("ERROR OPENING POLHEMUS!!\n");
        return;
    }
    write(device,"P",1);                /* request first packet */
}


/*--------------------------------------------------------------*/
/* Function polhemus_simul                                      */
/*                                                              */
/* PARAMETERS:                                                  */
/*      void                                                    */
/*                                                              */
/* PROCESS:                                                     */
/*      This function is called once during each frame.  It     */
/*      reads the data packet from the Polhemus, grabs the      */
/*      azimuth value (z rotation), stuffs the new value in     */
/*      place of the mechanical cupola, and requests the next   */
/*      data packet from the Polhemus.                          */
/*                                                               */
/* RETURNS:                                                      */
/*      void                                                     */
/*---------------------------------------------------------------*/

void polhemus_simul()
{
    read(device, temp_buffer, PACKET_SIZE);              /* read packet */
    sscanf(&(temp_buffer[3]), "%f", &azimuth);   /* azim first in AER packet */

    cupola_cws_new_value((azimuth/180.0)*1.195);         /* emulate the pot */
/*      dividing azimuth by 180.0 scales to [-1.0,1.0]                   */
/*      multiplying by 1.195 stretches this range to ~ [-1.2,1.2]        */
/*      This new value gives full circle capability, avoiding the        */
/*      'dead spot' that the mechanical cupola has.                      */

    write(device,"P",1);                 /* request next packet */
}



/*---------------------------------------------------------------*/
/* Function polhemus_exit                                        */
/*                                                               */
/* PARAMETERS:                                                   */
/*      void                                                     */
/*                                                               */
/* PROCESS:                                                      */
/*      This function is called once when finished to shutdown   */
/*      the Polhemus and restore the port to its original        */
/*      configuration.                                           */
/*                                                               */
/* RETURNS:                                                      */
/*      void                                                     */
/*---------------------------------------------------------------*/

void polhemus_exit()
{
    close_polhemus();
}



/*-------------------*/
/* PRIVATE functions */
/*-------------------*/

/*---------------------------------------------------------------*/
/* Function open_polhemus                                        */
/*                                                               */
/* PARAMETERS:                                                   */
/*      char *device_name - the tty port to open                 */
/*                                                               */
/* PROCESS:                                                      */
/*      This function opens the port, storing the termio data    */
/*      so that it can be restored later in the close routine.   */
/*      It also sets the Polhemus to ASCII mode, and sets the    */
/*      Polhemus data packet to azimuth/elevation/roll.          */
/*                                                               */
/* RETURNS:                                                      */
/*      on success, returns 1                                    */
/*      on failure, returns 0                                    */
/*---------------------------------------------------------------*/

int open_polhemus( device_name )
    char *device_name;
```

```
{
    strcpy(temp_buffer, device_name);
    if ((device = open(temp_buffer, O_RDWR)) < 0) {
        printf("ERROR in open_polhemus:  ");
        printf("open failed on %s\n", temp_buffer);
        return(0);
    }

    if (ioctl(device, TCGETA, &old_port_control) < 0) {
        printf("ERROR in open_polhemus:  ");
        printf("getting old port control information failed\n");
        return(0);
    }
    copy_termio(&old_port_control, &polhemus_control);

    polhemus_control.c_iflag = IXOFF | IXON | IGNBRK;
    polhemus_control.c_oflag = 0;
    polhemus_control.c_cflag = B19200 | CS8 | CREAD;
    polhemus_control.c_lflag = 0;
    polhemus_control.c_line  = 0;
    polhemus_control.c_cc[VMIN]  = PACKET_SIZE;  /* sizeof(DATA_PACKET_AER) */
    polhemus_control.c_cc[VTIME] = 0;

    if (ioctl(device, TCSETA, &polhemus_control) < 0) {
        printf("ERROR in open_polhemus:  ");
        printf("setting polhemus control information failed\n");
        return(0);
    }

    write(device,"F",1);                        /* set ASCII */
    write(device,"O4\r",3);                     /* set DATA_PACKET_AER */

    return(1);          /* everything must have worked to get this far */
}


/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* Function close_polhemus                                      */
/*                                                              */
/* PARAMETERS:                                                  */
/*      void                                                    */
/*                                                              */
/* PROCESS:                                                     */
/*      This function restores the port to its original         */
/*      configuration.                                          */
/*                                                              */
/* RETURNS:                                                     */
/*      void                                                    */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/

int close_polhemus() {

    if (ioctl(device, TCSETA, old_port_control) < 0) {
        printf("ERROR in close_polhemus:  ");
        printf("reset old port control info failed\n");
        return(0);
    }
    return(1);
}


/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* Function copy_termio                                         */
/*                                                              */
/* PARAMETERS:                                                  */
/*      struct termio source, dest - port configuration data    */
/*                                                              */
/* PROCESS:                                                     */
/*      This function copies the termio data from one structure */
/*      to another.                                             */
/*                                                              */
/* RETURNS:                                                     */
/*      void                                                    */
/*--------------------------------------------------------------*/

copy_termio( source, dest )
    struct termio *source, *dest;
{
    dest->c_iflag = source->c_iflag;
    dest->c_oflag = source->c_oflag;
    dest->c_cflag = source->c_cflag;
    dest->c_lflag = source->c_lflag;
    dest->c_line  = source->c_line;
    strncpy(dest->c_cc, source->c_cc, NCC);
}
```

```
#define VERSION "6.????"

/*****************************************************************
 * SYSTEM NAME: m1                                               *
 * FILE:        m1_main.c                                        *
 * AUTHORS:     David Epstein                                    *
 *              Joe Marks                                        *
 *              James Chung                                      *
 *              Art Pope                                         *
 *              John Morrison                                    *
 *              Alan Dickens                                     *
 *              Brian O'Toole                                    *
 *              Dan Van Hook                                     *
 *              Carol Chiang                                     *
 *              Maureen Saffi                                    *
 *                                                               *
 * SIMNET simulation of M-1 Abrams Main Battle Tank.             *
 *                                                               *
 * Copyright (c) 1988, 1989, 1990 BBN Systems and Technologies   *
 * All rights reserved.                                          *
 *                                                               *
 *****************************************************************/

#include "stdio.h"
#include "ctype.h"
#include "signal.h"
#include "sys/mpadvise.h"
#include "sim_dfns.h"
#include "sim_macros.h"
#include "sim_types.h"

#include "mass_stdc.h"
#include "dgi_stdg.h"
#include "sim_cig_if.h"

#include "pro_assoc.h"
#include "pro_sim.h"
#include "status.h"
#include "status_m1.h"
#include "veh_type.h"

#include "fifo_dfn.h"
#include "fifo.h"
#include "bigwheel.h"
#include "libterrain.h"
#include "libkin.h"
#include "libfail.h"
#include "libcig.h"
#include "libmsg.h"
#include "bbd.h"
#include "libhull.h"
#include "libidc.h"
#include "libmain.h"
#include "libmem.h"
#include "libnetwork.h"
#include "librepair.h"
#include "librva.h"
#include "libsusp.h"
#include "libturret.h"
#include "libsound.h"
#include "libmap.h"

#include "m1_ammo.h"
#include "m1_bcs.h"
#include "m1_cntrl.h"
#include "m1_cupola.h"
```

```
#include "m1_dtrain.h"
#include "m1_elecsys.h"
#include "m1_failure.h"
#include "m1_fuelsys.h"
#include "m1_firectl.h"
#include "m1_weapons.h"
#include "m1_handles.h"
#include "m1_hydrsys.h"
#include "m1_keybrd.h"
#include "m1_laser.h"
#include "m1_main.h"
#include "m1_meter.h"
#include "m1_polhemus.h"
#include "m1_pots.h"
#include "m1_repair.h"
#include "m1_resupp.h"
#include "m1_sound.h"
#include "m1_turret.h"
#include "m1_vision.h"
#include "m1_status.h"
#include "m1_thermal.h"

#include "timers.h"
#include "dtad.h"
#include "status.h"
#include "cmc.h"
#include "cmc_timer.h"
#include "cmc_status.h"
#include "ser_status.h"

#define PARS_FILE       "/simnet/vehicle/m1/data/m1_pars.d"
/*
#define CONFIG_FILE     "/simnet/vehicle/m1/data/m1_vconfig.d"
#define VEH_MAP_FILE    "/simnet/data/veh_map.d"
#define ASID_MAP_FILE   "/simnet/data/asid.d"
*/

BOOLEAN debug = FALSE;
BOOLEAN print_overruns = FALSE;

static BOOLEAN polhemus_flag;
static BOOLEAN guise_override = FALSE;
static int guise_to_use;

#define MESSAGE         "Eat at Mama Luigi's"
#define BOLD_FLASH      "[1;5m"
#define NORMAL          "[0m"

/* from the '-p' switch */
static ActivateRequestVariant init_activ, *initial_activation = NULL;

#ifndef SIMBFLY

void exit();

#endif

void print_help (progname)
char *progname;
{
    printf ("Usage: %s \n", progname);
    printf ("switches...\n");
    printf ("\t-a(symmetric buffers: receive send)\n");
    printf ("\t-c(upola and loader's periscope controllable by keyboard)\n");
    printf ("\t-C(atc Hardware used)\n");
    printf ("\t-d(ebugging on)\n");
    printf ("\t-D(ebugging for static vehicles on)\n");
    printf ("\t-e(thernet off)\n");
    printf ("\t-E(xercise id)\n");
    printf ("\t-F(ail debug on)\n");
    printf ("\t-g(raphics off)\n");
    printf ("\t-G(uise to use instead of US_M1, in hex)\n");
    printf ("\t-h(elp)\n");
    printf ("\t-i(ndicate vpkt sent)\n");
    printf ("\t-k(eyboard on)\n");
    printf ("\t-m(essages for equipment status not printed)\n");
    printf ("\t-n(etwork verbose mode)\n");
    printf ("\t-N(ight vision on)\n");
    printf ("\t-o(verrun printing)\n");
    printf ("\t-p(osition) initial_X initial_Y initial_heading\n");
    printf ("\t-P(riority list debugging on)\n");
    printf ("\t-s(ound off)\n");
    printf ("\t-t(errain database) database_name\n");
    printf ("\t-T ded database) database_name\n");
    printf ("\t-v(erbose mode on)\n");
    printf ("\t-?(help)\n");
}

void
print_veh_logo ()
{
    printf ("%c[H%c[J", '\033', '\033');                /* clear screen */
printf ("                                                     | \n");
printf ("                                                     | \n");
printf ("                                                     | \n");
printf ("                                                     | \n");
printf ("                                  --------+          | \n");
printf ("                                          + + +      | \n");
printf ("                                  + + +   + +        | \n");
printf ("                     /+ + + + + + + + + + + + + + + + + + + \n");
printf ("=====================          %c%s%s%c%s         + \n",0x1b,BOLD_FLASH,
printf ("                     \\+                                  + \n");
printf ("                + + + + + + +                      + + + + + + + + \n");
printf ("                +                                            +   + \n");
printf ("                  \\                                        +   / \n");
printf ("                    \\  - - - - - - - - - - - - - - - - - - - / \n");
printf ("                      \\   ( )  ( )  ( )  ( )  ( )  ( )     / \n");
printf ("                         ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ \n");
printf ("\n");
printf("              -------------------------------------------------\n");
printf("                         SIMNET M1 SIMULATION  V%s\n", VERSION);
printf("                  Copyright (c) 1990 BBN Systems and Technologies\n");
printf("                            All rights reserved.\n");
printf("              -------------------------------------------------\n");

#ifndef SIMBFLY

    sleep (5);

#else

    Sleep(8000);

#endif

    printf ("%c[H%c[J", '\033', '\033');                /* clear screen */
}

void veh_spec_startup()
{
    extern void rtc_init_clock();
    rtc_init_clock();
/*    main_read_pars_file (PARS_FILE);*/
/*
    vehicle_type is in activate.  We DO need to set simulator type, however
    network_set_vehicle_type ((int) VEH_MAIN_BATTLE_TANK);
*/
    network_set_simulator_type (simulator_SIMNET_M1);
    use_cig_reconfig_startup ();
    cig_set_view_config_file (get_vconfig_file1 ());
    map_vehicle_file_read (get_veh_map_file ());
    map_read_asid_file (get_asid_map_file ());
    keyboard_init ();
/*    weapons_download_ballistics_tables();*/

/* this function should move out of vehicle specific code */
/*    map_file_read("/simnet/data/trial_map.d");*/
    map_file_read(get_ammo_map_file ());

    failure_init ();            /* initialize damage tables      */
    map_get_damage_files();     /* must be after failure_init    */
}

void veh_spec_idle()
{
    status_simul ();
    keyboard_simul ();
    io_simul_idle ();

    if (initial_activation != NULL)
    {
        process_activate_request (initial_activation,(SimulationAddress *) 0,
                0, network_get_exercise_id ());
        initial_activation = NULL;
    }
}

void veh_spec_init()
{
/* Order dependent initialization here. */
    cupola_init ();             /* must be before controls */
    sound_init ();              /* before controls, after idcs */
    status_preset ();           /* after idc init */
        /* does ammo_init still need to be before controls or */
        /* is this one of those historical comments?          */
        /* I am assuming the latter. -CJC 3/16/90 */
/*    ammo_init ();             /* must be before controls */
    controls_fsm_init ();

    resupply_init ();
    meter_init ();              /* must be before electsys */
    electsys_init ();           /* this must be after controls_init */
    hydraulic_init ();
    firectl_init ();            /* this must be before laser_init */
/*    fuel_init ();*/
    drivetrain_init ();
    handles_init ();
    laser_init ();
    bcs_init ();
    weapons_init ();
    vision_restore_all_blocks ();

    controls_edge_init ();
    app_init ();
    thermal_init();
    config_pos_init2( kinematics_get_o_to_h(veh_kinematics),
                     kinematics_get_w_to_h(veh_kinematics) );
    cig_init_ctr ();
}

void veh_spec_simulate()
{
    status_simul ();
#if defined(SIMBFLY)
    {
    /* ### don't count printing of stats against simulation */
        long    start, end;

        start = rtc;            /* END_FRAGMENT(60) */
        keyboard_simul ();
        end = rtc;
        bbd_bit_start[60] += end - start;/* START_FRAGMENT(60) */
    }
#else
    keyboard_simul ();
#endif
    sound_simul ();             /* should be first */
    controls_simul ();
    handles_simul ();           /* must be before kinemat simuls */
    ammo_simul ();
    resupply_simul ();
    electsys_simul ();
    hydraulic_simul ();
    fuel_simul ();
    drivetrain_simul ();
    bcs_simul ();
    weapons_simul ();
    if (polhemus_flag)
        polhemus_simul();
    cupola_simul ();
    thermal_simul ();
}

void veh_spec_stop()
{
    idc_init ();
    sound_init ();
    vision_break_all_blocks ();
}

void veh_spec_exit()
{
    int     num_ticks;

    keyboard_exit_gracefully ();
    printf ("Elapsed pseudo time %lf secs.\n", timers_get_current_time ());
    printf ("Elapsed time        %d ticks.\n", timers_get_current_tick ());
    printf ("Elapsed real time   %lf secs.\n",
            (timers_elapsed_milliseconds () / 1000.0));
    if ((num_ticks = timers_get_current_tick ()) != 0)
    {
        printf ("Average frame time  %lf msecs.\n",
                ((REAL) timers_elapsed_milliseconds () / (REAL) num_ticks));
    }
    network_print_statistics ();

    net_close (net_handle);
}

main (argc, argv)
int argc;
char *argv [];
{
```

```
int     i;

signal (SIGINT, (PFI)exit_gracefully);
signal (SIGTERM, (PFI)exit_gracefully);

enter_gracefully ();                        /* print copyright banner */

network_set_exercise_id (1);

printf ("ALERT: Initial buffer transfer size is 512 x 512 !!\n");
main_read_pars_file (PARS_FILE);

for (i = 1; i < argc; i++)
{
    switch (argv[i][0])
    {
        case '-':
            switch (argv[i][1])
            {
                case 'a':
                    set_request_receive_size (atoi( argv[++i] ));
                    set_request_send_size (atoi (argv[++i]));
                    set_assymetric_on ();
                    break;
                case 'A':
                    ammo_enable_autoloader();
                    break;
                case 'c':
                    keyboard_use_cupola ();
                    break;
                case 'C':
                    set_catc_mode ();
                    printf ("using catc89 Hardware\n");
                    break;
                case 'd':
                    debug = 1;
                    printf ("Debugging is now on...\n");
                    break;
                case 'D':
                    use_static_debug (1);
                    printf ("debugging for static vehicles on...\n");
                    break;
                case 'e':
                    network_dont_really_open_up_ethernet ();
                    break;
                case 'E':
                    network_set_exercise_id ((ExerciseID)(atoi(argv[++i])));
                    break;
                case 'F':
                    cfail_debug_on();
                    break;
                case 'g':
                    cig_not_using_graphics ();
                    break;
                case 'G':
                    guise_override = TRUE;
                    sscanf (argv[++i], "%x", &guise_to_use);
                    printf("ml using guise 0x%08x\n", guise_to_use);
                    break;
                case 'h':
                case '?':
                    print_help (argv[0]);
                    exit (-1);
                case 'k':
                    keyboard_really_use ();
                    break;
```

```
case 'l':
    /*laser_enable_dazzler();*/
    printf("no dazzling laser available\n");
    exit(-1);
case 'm':
    disable_status_printing();
    break;
case 'n':
    v_pkt_verbose_mode ();
    break;
case 'N':
    printf ("Night vision enabled\n");
    vision_set_otw_night_vision ();
    break;
case 'o':
    print_overruns = TRUE;
    printf ("Printing of overruns is now on...\n");
    break;
case 'p':
    {
    float initial_heading;  /* degrees      */

    SIMNET_M1_Status *status;
    GroundVehicleSubsystems *gp;
    OrganizationalUnit *unit;
    unsigned int status_bits;

    init_activ.reason = activateReasonOther;
    init_activ.vehicleClass = vehicleClassTank;

    /* VehicleID is set at net_init time, so the
       one in the activate pkt is ignored... */

    unit = & init_activ.unit;
    unit->force = distinguishedForceID;
    unit->organizationType = organizationArmy;

    unit->hierarchy[0].unitType = unitTypeArmy;
    unit->hierarchy[0].unitNumber = 0;
    unit->hierarchy[1].unitType = unitTypeCorps;
    unit->hierarchy[1].unitNumber = 0;
    unit->hierarchy[2].unitType = unitTypeDivision;
    unit->hierarchy[2].unitNumber = 0;
    unit->hierarchy[3].unitType = unitTypeBrigade;
    unit->hierarchy[3].unitNumber = 0;
    unit->hierarchy[4].unitType = unitTypeBattalion;
    unit->hierarchy[4].unitNumber = 0;
    unit->hierarchy[5].unitType = unitTypeCompany;
    unit->hierarchy[5].unitNumber = 0;
    unit->hierarchy[6].unitType = unitTypePlatoon;
    unit->hierarchy[6].unitNumber = 0;
    unit->hierarchy[7].unitType = unitTypeIrrelevant;
    unit->hierarchy[7].unitNumber = 0;
    unit->hierarchy[8].unitType = unitTypeSquad;
    unit->hierarchy[8].unitNumber = 0;

    init_activ.marking.characterSet = asciiCharacterSet;
    strcpy (init_activ.marking.text, "A10");

    if (guise_override)
    {
    init_activ.guises.distinguished = guise_to_use;
    init_activ.guises.other = guise_to_use;
    }
    else
    {
```

```
init_activ.guises.distinguished = vehicle_US_M1;
init_activ.guises.other = vehicle_US_M1;
}

init_activ.simulatedTime = 0;

strcpy (init_activ.terrain.terrainName,
        get_default_db_name ());
init_activ.terrain.terrainVersion =
        get_default_db_version ();

sscanf (argv[++i], "%lf", &(init_activ.location[X]));
sscanf (argv[++i], "%lf", &(init_activ.location[Y]));
sscanf (argv[++i], "%f", &initial_heading);

printf("Initializing tank @ (%lf, %lf) heading %f\n",
        init_activ.location[X],
        init_activ.location[Y],
        initial_heading);

init_activ.battleScheme = battleSchemeOther;
init_activ.onSurface = TRUE;

init_activ.status.vehicleType = vehicle_US_M1;

/* brand spanking new */
init_activ.status.odometer = 0; /* meters */
init_activ.status.age = 1; /* years */

init_activ.status.failures.category =
    groundVehicleSubsystems;
init_activ.status.failures.operationalSummary = 0;
init_activ.status.failures.mobilitySummary = 0;
init_activ.status.failures.firepowerSummary = 0;
init_activ.status.failures.communicationSummary = 0;
init_activ.status.failures.noncriticalSummary = 0;

gp = &init_activ.status.failures.subsystems.ground;

    /* initialize everything working */
status_bits = 0x0;
BCOPY ((char *) &status_bits,
        (char *) &gp -> motive[subsystemExists],
        sizeof (MotiveSubsystems));
BCOPY ((char *) &status_bits,
        (char *) &gp -> electronic[subsystemExists],
        sizeof (ElectronicSubsystems));
BCOPY ((char *) &status_bits,
        (char *) &gp -> power[subsystemExists],
        sizeof (PowerSubsystems));
BCOPY ((char *) &status_bits,
        (char *) &gp -> weapon[subsystemExists],
        sizeof (WeaponSubsystems));
BCOPY ((char *) &status_bits,
        (char *) &gp -> chassis[subsystemExists],
        sizeof (ChassisSubsystems));
BCOPY ((char *) &status_bits,
        (char *) &gp -> turret[subsystemExists],
        sizeof (TurretSubsystems));

status_bits = 0;
BCOPY ((char *) &status_bits,
        (char *) &gp -> motive[subsystemStatus],
        sizeof (MotiveSubsystems));
BCOPY ((char *) &status_bits,
        (char *) &gp -> electronic[subsystemStatus],
                sizeof (ElectronicSubsystems));
        BCOPY ((char *) &status_bits,
                (char *) &gp -> power[subsystemStatus],
                sizeof (PowerSubsystems));
        BCOPY ((char *) &status_bits,
                (char *) &gp -> weapon[subsystemStatus],
                sizeof (WeaponSubsystems));
        BCOPY ((char *) &status_bits,
                (char *) &gp -> chassis[subsystemStatus],
                sizeof (ChassisSubsystems));
        BCOPY ((char *) &status_bits,
                (char *) &gp -> turret[subsystemStatus],
                sizeof (TurretSubsystems));


        init_activ.status.specific.category = simnetM1Status;
        status = &init_activ.status.specific.specific.m1;

        status -> enginePower = 1.0;
        status -> battery = 24.0;
        status -> frontLeftFuel = 70.0;
        status -> frontRightFuel = 70.0;
        status -> rearFuel = 150.0;

        status -> apdsReadyAmmo = 12;
        status -> apdsSemiReadyAmmo = 11;
        status -> apdsHullTurretFloorAmmo = 6;
        status -> heatReadyAmmo = 10;
        status -> heatSemiReadyAmmo = 11;
        status -> heatHullTurretFloorAmmo = 5;

        init_activ.specific.tank.hullAzimuth =
            (unsigned long) (4294967295 -
            (unsigned long) (initial_heading *
                4294967295.0 / 360.0));
        init_activ.specific.tank.turretAzimuth = 0;


        initial_activation = &init_activ;

        break;
        }
    case 'P':
        rva_turn_debug_on();
        printf("turning priority list debugging on\n");
        break;
    case 's':
        sound_dont_use();
        break;
    case 't':
        if( !isalpha( argv[++i][0] ) )
            {
            printf("Cannot use invalid terraindatabaser: %s\n",
                    argv[i] );
            exit( 0 );
            }
        cig_use_database_override_named( argv[i] );
        break;
    case 'T':
        if( !isalpha( argv[++i][0] ) )
            {
            printf("Cannot use invalid terraindatabaser: %s\n",
                    argv[i] );
            exit( 0 );
            }
        set_ded_name( argv[i] );
```

```
                        break;
                    case 'v':
                        terrain_verbose_mode_on ();
                        break;
                    case '1':
                        set_cig_dev (1, atoi(argv[++i]));
#ifdef _GT_
                        his_cif_interface = atoi (argv[i]);
#endif
                        set_cig_mask (CIG_1);
                        break;
                    case '2':
                        set_cig_dev (2, atoi(argv[++i]));
                        set_cig_mask (CIG_2);
                        break;
                    case 'z':
                        {
                        double scale;
                        sscanf( argv[++i], "%lf", &scale);
                        printf("Using vehicle size scale of %lf\n", scale );
                        break;
                        }
                    default:
                        fprintf (stderr, "Unknown switch \"%s\"\n", argv[i]);
                        break;
                }
                break;
            default:
                fprintf (stderr, "unknown arg \"%s\"...\n", argv[i]);
                break;
        }
    }

    sim_state_startup();

    for(;;)
    {
        simulation_state_machine();
    }

    /*NOTREACHED*/                /* this keeps lint happy        */
    return (0);                   /* Exit gracefully, dummy up lint. */
}

void
reconstitute_vehicle ()
{
process_activate_request( &init_activ, (SimulationAddress *)0,
        (TransactionIdentifier) 0, network_get_exercise_id() );
}

void
set_polhemus_flag_true()
{
        polhemus_flag = TRUE;
}

void
set_polhemus_flag_false()
{
        polhemus_flag = FALSE;
}
```

```
/****************************************************************
 *                                                              *
 * FILE:          m1_ctl_npc.c                                  *
 * AUTHOR:        Brian O'Toole                                 *
 * MAINTAINER:    Brian O'Toole                                 *
 * HISTORY:       4/30/86  brian: Creation                      *
 *                9/29/87  brian: Added redistribute send       *
 *                12/06/89 fmh: Change to 36-bit munitions types: *
 *                              SABOT ==> M392A2, HEAT ==> M456A1 *
 *                7/2/91  mla (IST): added using_polhemus flag  *
 *                                and routines to manipulate it *
 *                                see note below                *
 *                                                              *
 * Copyright (c) 1986 BBN Laboratories, Inc.                    *
 * All rights reserved.                                         *
 *                                                              *
 ****************************************************************/

#include "stdio.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "sim_macros.h"

#include "mass_stdc.h"
#include "dgi_stdg.h"
#include "sim_cig_if.h"

#include "libcig.h"
#include "dtad.h"
#include "libidc.h"
#include "libidc_dfn.h"
#include "libmem.h"
#include "libmem_dfn.h"
#include "libnetwork.h"
#include "status.h"
#include "timers.h"
#include "timers_dfn.h"

#include "m1_ammo.h"
#include "m1_ammo_df.h"
#include "m1_ammo_mx.h"
#include "m1_ammo_pn.h"
#include "m1_cntrl.h"
#include "m1_ctl_df.h"
#include "m1_cupola.h"
#include "m1_driv_pn.h"
#include "m1_dtrain.h"
#include "m1_gunn_mx.h"
#include "m1_comm_mx.h"
#include "m1_load_mx.h"
#include "m1_hydrsys.h"
#include "m1_idc.h"
#include "m1_pots.h"
#include "m1_repair.h"
#include "m1_status.h"
#include "m1_thermal.h"
#include "m1_tmrs_df.h"
#include "m1_tracks.h"
#include "m1_turr_pn.h"
#include "m1_turret.h"
#include "m1_vision.h"

#include "mun_type.h"

#define INVERT_DELAY 15
```

```
/* ammo rack internals */
static int      apds_translations [N_READY] =
{
                            APDS_RDY04, APDS_RDY01,
    APDS_RDY09, APDS_RDY07, APDS_RDY05, APDS_RDY02,
    APDS_RDY10, APDS_RDY08, APDS_RDY06, APDS_RDY03,
    APDS_RDY20, APDS_RDY17, APDS_RDY14, APDS_RDY11,
    APDS_RDY21, APDS_RDY18, APDS_RDY15, APDS_RDY12,
    APDS_RDY22, APDS_RDY19, APDS_RDY16, APDS_RDY13
};


static int      heat_translations [N_READY] =
{
                            HEAT_RDY04, HEAT_RDY01,
    HEAT_RDY09, HEAT_RDY07, HEAT_RDY05, HEAT_RDY02,
    HEAT_RDY10, HEAT_RDY08, HEAT_RDY06, HEAT_RDY03,
    HEAT_RDY20, HEAT_RDY17, HEAT_RDY14, HEAT_RDY11,
    HEAT_RDY21, HEAT_RDY18, HEAT_RDY15, HEAT_RDY12,
    HEAT_RDY22, HEAT_RDY19, HEAT_RDY16, HEAT_RDY13
};


static int      select_translations [N_READY] =
{
                          ASELECT04, ASELECT01,
    ASELECT09, ASELECT07, ASELECT05, ASELECT02,
    ASELECT10, ASELECT08, ASELECT06, ASELECT03,
    ASELECT20, ASELECT17, ASELECT14, ASELECT11,
    ASELECT21, ASELECT18, ASELECT15, ASELECT12,
    ASELECT22, ASELECT19, ASELECT16, ASELECT13
};


static int      turret_ref_translations [TURRET_REF_NUM_SECTORS] =
{
    TURRET_TO_HULL_030, TURRET_TO_HULL_060, TURRET_TO_HULL_090,
    TURRET_TO_HULL_120, TURRET_TO_HULL_150, TURRET_TO_HULL_180,
    TURRET_TO_HULL_210, TURRET_TO_HULL_240, TURRET_TO_HULL_270,
    TURRET_TO_HULL_300, TURRET_TO_HULL_330, TURRET_TO_HULL_360
};


static REAL     real_service_brake_val;
static REAL     real_comm_weap_val;
static REAL     real_load_peri_val;
static int      hex_service_brake_val;
static int      hex_comm_weap_val;
static int      hex_load_peri_val;
static char     gps_mag_val;
static char     breech_ready;
static char     ejection_guard_val;
static char     ammo_transfer_status;
static char     grid_azimuth_status;
static int      fuel_flash_count;
static char     fuel_flash_status;
static int      odometer_timer_number;
static char     cupola_up_down_val;
static char     binoculars_on_off_val;
static char     lpscope_up_down_val;
static char     thermal_shutter_val;

static BOOLEAN  using_polhemus = FALSE;         /* added for polhemus */
```

```
                                                                /* integration          */
                                                                /* July 2, 1991         */
                                                                /* M Altman (IST)       */
                                                                /* see note below       */

/* no power routines */
static void     controls_parking_brake_check ();
static void     controls_service_brake_check ();
static void     controls_mag_check ();
static void     controls_ejection_guard_check ();
static void     controls_breech_check ();
static void     controls_breech_unload_check ();
static void     controls_breech_ready_check ();
static void     controls_ammo_transfer_check ();
static void     controls_knee_switch_check ();
static void     controls_ammo_tube_check ();
static void     controls_commander_weapon_station_check ();
static void     controls_loader_periscope_check ();
static void     controls_grid_azimuth_check ();
static void     controls_fuel_flash_check ();
static void     controls_odometer_check ();
static void     controls_cupola_up_down_check ();
static void     controls_binoculars_on_off_check ();
static void     controls_lpscope_up_down_check ();
static void     controls_thermal_shutter_check ();

/* init routines */
static void     controls_service_brake_init ();
static void     controls_mag_init ();
static void     controls_ejection_guard_init ();
static void     controls_ammo_transfer_init ();
static void     controls_commander_weapon_station_init ();
static void     controls_loader_periscope_init ();
static void     controls_cupola_up_down_init ();
static void     controls_binoculars_on_off_init ();
static void     controls_lpscope_up_down_init ();
static void     controls_thermal_shutter_init ();

static void     controls_transfer_semi_heat ();
static void     controls_transfer_semi_apds ();
static void     controls_transfer_hull_heat ();
static void     controls_transfer_hull_apds ();
static void     controls_transfer_no_transfer ();
static void     controls_transfer_redist_send ();
static void     controls_transfer_redist_recv ();
static void     controls_service_brake_exit ();
static void     controls_odometer_exit ();
static void     controls_cupola_up_down_exit ();
static void     controls_binoculars_on_off_exit ();
static void     controls_lpscope_up_down_exit ();
static void     controls_fuel_flash ();
static void     controls_fuel_unflash ();
static void     controls_fuel_restore ();


/*---------------------------------------------------------------------------*/
/*      The using_polhemus flag and these two routines to manipulate it */
/*      were added to prevent a read of the TC's cupola pot while the   */
/*      polhemus is active.  If the cupola pot is also read while the   */
/*      polhemus is active, there is an oscillation as the polhemus and */
/*      the pot fight over the 'correct' value.                         */
/*                                                                      */
/*      July 2, 1991 - M Altman (IST)                                   */
/*---------------------------------------------------------------------------*/
```

```
void controls_set_using_polhemus_true() {
    using_polhemus = TRUE;
}

void controls_set_using_polhemus_false() {
    using_polhemus = FALSE;
}
/*----------------------------------------------------------------------*/


void controls_npc_init ()
{
    real_service_brake_val = 0.0;
    real_comm_weap_val = 0.0;
    real_load_peri_val = 0.0;
    hex_service_brake_val = 0;
    hex_comm_weap_val = 0;
    hex_load_peri_val = 0;
    gps_mag_val = GN_3X_VAL;
    breech_ready = OFF;
    ejection_guard_val = OFF;
    ammo_transfer_status = NO_TRANSFER_VAL;
    grid_azimuth_status = OFF;
    fuel_flash_count = 0;
    fuel_flash_status = OFF;
    odometer_timer_number = timers_set_null_timer ();
    cupola_up_down_val = CM_CENTER_VAL;
    binoculars_on_off_val = OFF;
    lpscope_up_down_val = LD_CENTER_VAL;

    controls_service_brake_init ();
    controls_mag_init ();
    controls_ejection_guard_init ();
    controls_ammo_transfer_init ();
    controls_commander_weapon_station_init ();
    controls_loader_periscope_init ();
    controls_cupola_up_down_init ();
    controls_binoculars_on_off_init ();
    controls_lpscope_up_down_init ();
    controls_thermal_shutter_init ();
}



void controls_no_power_routines ()
{
    controls_parking_brake_check ();
    controls_service_brake_check ();
    controls_mag_check ();
    controls_ejection_guard_check ();
    controls_breech_check ();
    controls_breech_unload_check ();
    controls_breech_ready_check ();
    controls_ammo_transfer_check ();
    controls_knee_switch_check ();
    controls_ammo_tube_check ();
    controls_commander_weapon_station_check ();
    controls_loader_periscope_check ();
    controls_grid_azimuth_check ();
    controls_fuel_flash_check ();
    controls_odometer_check ();
    controls_cupola_up_down_check ();
    controls_binoculars_on_off_check ();
    controls_lpscope_up_down_check ();
    controls_thermal_shutter_check ();
}
```

```
static void controls_parking_brake_check ()

{
    if (idc_values [DR_P_BRAKE_SET])
    {
        idc_values [DR_P_BRAKE_SET] = OFF;
        if (hydraulic_parking_brake_on_request ())
        {
            drivetrain_set_parking_brake ();
            controls_set_parking_brake ();
        }
    }

    if (idc_values [DR_P_BRAKE_RELEASE])
    {
        idc_values [DR_P_BRAKE_RELEASE] = OFF;
        drivetrain_release_parking_brake ();
        controls_release_parking_brake ();
    }
}


static void controls_service_brake_check ()
{
    int temp;

    if ((temp = potval(DR_S_BRAKE)) != hex_service_brake_val)
    {
        real_service_brake_val =
                pots_service_brake_real (hex_service_brake_val = temp);
        drivetrain_set_service_brake (real_service_brake_val);
    }
}


static void controls_service_brake_init ()
{
    hex_service_brake_val = potval(DR_S_BRAKE);
    real_service_brake_val = pots_service_brake_real (hex_service_brake_val);
    drivetrain_set_service_brake (real_service_brake_val);
}


static void controls_service_brake_exit ()
{
    drivetrain_set_service_brake (0.0);
}


static void controls_mag_check ()
{
    char temp_3x, temp_10x;

    temp_3x  = idc_values [GN_GPS_MAG_3X];
    temp_10x = idc_values [GN_GPS_MAG_10X];

    if (thermal_view_on())
    {
        return;
    }
    if ((temp_3x) &&
        (gps_mag_val != GN_3X_VAL) &&
        (! temp_10x))
    {
        gps_mag_val = GN_3X_VAL;
        cig_gps_mag_3x ();
    }

    else if ((temp_10x) &&
        (gps_mag_val != GN_10X_VAL) &&
        (! temp_3x))
    {
        gps_mag_val = GN_10X_VAL;
        cig_gps_mag_10x ();
    }
}


static void controls_mag_init ()
{
    char temp_3x, temp_10x;

    temp_3x  = idc_values [GN_GPS_MAG_3X];
    temp_10x = idc_values [GN_GPS_MAG_10X];

    if ((temp_3x) &&
        (! temp_10x))
    {
        printf("mag_init: Initialize to Low MAG ******************\n");
        gps_mag_val = GN_3X_VAL;
        cig_gps_mag_3x ();
    }

    else if ((temp_10x) &&
        (! temp_3x))
    {
        printf("mag_init: Initialize to Hi MAG ******************\n");
        gps_mag_val = GN_10X_VAL;
        cig_gps_mag_10x ();
    }

    else
    {
        printf("mag_init: Mag switch in weird state **********\n");
    }
}

char get_non_thermal_mag()
{
    return(gps_mag_val);
}


static void controls_ejection_guard_check ()
{
    char temp;

    if ((temp = idc_values [LD_EJECTION_GUARD]) != ejection_guard_val)
    {
        switch (ejection_guard_val = temp)
        {
            case ON:
                ammo_ejection_guard_armed ();
                controls_ejection_guard_armed ();
                break;
            case OFF:
                ammo_ejection_guard_safe ();
                controls_ejection_guard_safe ();
                break;
            default:
                fprintf (stderr, "CONTROLS: controls_ejection_guard_check: impossib
                nprintf ("CONTROLS: controls_ejection_guard_check: impossible eject
                break;
        }
    }
}


static void controls_ejection_guard_init ()
{
    switch (ejection_guard_val = idc_values [LD_EJECTION_GUARD])
    {
        case ON:
            ammo_ejection_guard_armed ();
            controls_ejection_guard_armed ();
            break;
        case OFF:
            ammo_ejection_guard_safe ();
            controls_ejection_guard_safe ();
            break;
        default:
            fprintf (stderr, "CONTROLS: controls_ejection_guard_init: impossible ej
            nprintf ("CONTROLS: controls_ejection_guard_init: impossible ejection g
            break;
    }
}


static void controls_breech_check ()
{
    if (idc_values [LD_SHELL_LOADED_PB])
    {
        ammo_breech_pushed ();
        idc_values [LD_SHELL_LOADED_PB] = OFF;
    }
}


static void controls_breech_unload_check ()
{
    if (idc_values [LD_UNLOAD_BREECH_PB])
    {
        ammo_breech_unload_pushed ();
        idc_values [LD_UNLOAD_BREECH_PB] = OFF;
    }
}


static void controls_knee_switch_check ()
{
    switch (idc_values [LD_KNEE_SWITCH])
    {
        case ON:
            ammo_knee_switch_on ();
            break;
        case OFF:
            ammo_knee_switch_off ();
            break;
        default:
            fprintf (stderr, "CONTROLS: controls_knee_switch_check: Impossible knee
            nprintf ("CONTROLS: controls_knee_switch_check: Impossible knee switch
            break;
    }
}


void controls_ejection_guard_armed ()
{
    idc_output_set_cond  (((controls_power_status () ==
                         CONTROLS_STATE_TURRET_POWER_ON) &&
                        (controls_electsys_status ()) &&
                        (! controls_failure_status ())),
                       LD_MAIN_ARMED_L, OUTPUT_ON);
    idc_output_set_cond (! controls_commander_panel_status (),
                       LD_MAIN_SAFE_L, OUTPUT_OFF);
}


void controls_ejection_guard_safe ()
{
    idc_output_set_cond (! controls_commander_panel_status (),
                       LD_MAIN_ARMED_L, OUTPUT_OFF);
    idc_output_set_cond  (((controls_power_status () ==
                         CONTROLS_STATE_TURRET_POWER_ON) &&
                        (controls_electsys_status ()) &&
                        (! controls_failure_status ())),
                       LD_MAIN_SAFE_L, OUTPUT_ON);
}


void controls_set_parking_brake ()
{
    if (! idc_values [DR_BRAKE_L])
    {
        idc_output_set_cond (((controls_power_status () !=
                            CONTROLS_STATE_NO_POWER) &&
                           (controls_electsys_status ()) &&
                           (! controls_failure_status ())),
                          DR_BRAKE_L, OUTPUT_ON);
        idc_output_set_cond (((controls_power_status () !=
                            CONTROLS_STATE_NO_POWER) &&
                           (controls_electsys_status ()) &&
                           (! controls_failure_status ())),
                          DR_MASTER_WARNING_L, OUTPUT_ON);
    }
}

void controls_release_parking_brake ()
{
    if (idc_values [DR_BRAKE_L])
    {
        idc_output_set_cond (! controls_driver_panel_status (),
                           DR_BRAKE_L, OUTPUT_OFF);
        controls_warning_lamp_off_check ();
    }
}
```

```
static void controls_breech_ready_check ()
{
    char temp;

    if ((temp = ammo_breech_ready ()) != breech_ready)
    {
        switch (breech_ready = temp)
        {
            case ON:
                idc_output_set_cond (! controls_failure_status (),
                                  LD_BREECH_READY_L, OUTPUT_ON);
                break;
            case OFF:
                idc_output_set_cond (! controls_commander_panel_status (),
                                   LD_BREECH_READY_L, OUTPUT_OFF);
                break;
            default:
                fprintf (stderr, "CONTROLS: controls_breech_ready_check: Impossible
                nprintf ("CONTROLS: controls_breech_ready_check: Impossible breech_
                break;
        }
    }
}


static void controls_ammo_transfer_check ()
{
    char temp_semi_heat, temp_semi_apds, temp_hull_heat, temp_hull_apds;
    char temp_redist_send, temp_redist_recv;

    temp_semi_heat   = idc_values [SEMI_HEAT];
    temp_semi_apds  = idc_values [SEMI_APDS];
    temp_hull_heat   = idc_values [HULL_HEAT];
    temp_hull_apds  = idc_values [HULL_APDS];
    temp_redist_send = idc_values [REDIST_SEND];
    temp_redist_recv = idc_values [REDIST_RECV];

    if ((temp_semi_heat) &&
        (ammo_transfer_status != SEMI_HEAT_VAL) &&
        (! temp_semi_apds) &&
        (! temp_hull_heat) &&
        (! temp_hull_apds) &&
        (! temp_redist_send) &&
        (! temp_redist_recv))
    {
        controls_transfer_semi_heat ();
        ammo_transfer_semi_heat ();
        ammo_transfer_status = SEMI_HEAT_VAL;
    }

    else if ((temp_semi_apds) &&
        (ammo_transfer_status != SEMI_APDS_VAL) &&
        (! temp_semi_heat) &&
        (! temp_hull_heat) &&
        (! temp_hull_apds) &&
        (! temp_redist_send) &&
        (! temp_redist_recv))
    {
        controls_transfer_semi_apds ();
        ammo_transfer_semi_apds ();
        ammo_transfer_status = SEMI_APDS_VAL;
    }

    else if ((temp_hull_heat) &&
        (ammo_transfer_status != HULL_HEAT_VAL) &&
        (! temp_semi_heat) &&
        (! temp_semi_apds) &&
        (! temp_hull_apds) &&
        (! temp_redist_send) &&
        (! temp_redist_recv))
    {
        controls_transfer_hull_heat ();
        ammo_transfer_hull_heat ();
        ammo_transfer_status = HULL_HEAT_VAL;
    }

    else if ((temp_hull_apds) &&
        (ammo_transfer_status != HULL_APDS_VAL) &&
        (! temp_semi_heat) &&
        (! temp_semi_apds) &&
        (! temp_hull_heat) &&
        (! temp_redist_send) &&
        (! temp_redist_recv))
    {
        controls_transfer_hull_apds ();
        ammo_transfer_hull_apds ();
        ammo_transfer_status = HULL_APDS_VAL;
    }

    else if ((! temp_semi_heat) &&
        (ammo_transfer_status != NO_TRANSFER_VAL) &&
        (! temp_semi_apds) &&
        (! temp_hull_heat) &&
        (! temp_hull_apds) &&
        (! temp_redist_send) &&
        (! temp_redist_recv))
    {
        controls_transfer_no_transfer ();
        ammo_transfer_no_transfer ();
        ammo_transfer_status = NO_TRANSFER_VAL;
    }

    else if ((temp_redist_send) &&
        (ammo_transfer_status != REDIST_SEND_VAL) &&
        (! temp_semi_heat) &&
        (! temp_semi_apds) &&
        (! temp_hull_heat) &&
        (! temp_hull_apds) &&
        (! temp_redist_recv))
    {
        controls_transfer_redist_send ();
        ammo_transfer_redist_send ();
        ammo_transfer_status = REDIST_SEND_VAL;
    }

    else if ((temp_redist_recv) &&
        (ammo_transfer_status != REDIST_RECV_VAL) &&
        (! temp_semi_heat) &&
        (! temp_semi_apds) &&
        (! temp_hull_heat) &&
        (! temp_hull_apds) &&
        (! temp_redist_send))
    {
        controls_transfer_redist_recv ();
        ammo_transfer_redist_recv ();
        ammo_transfer_status = REDIST_RECV_VAL;
    }
}


static void controls_ammo_transfer_init ()
```

```
{
    if (idc_values [SEMI_HEAT])
    {
        controls_transfer_semi_heat ();
        ammo_transfer_semi_heat ();
        ammo_transfer_status = SEMI_HEAT_VAL;
    }

    else if (idc_values [SEMI_APDS])
    {
        controls_transfer_semi_apds ();
        ammo_transfer_semi_apds ();
        ammo_transfer_status = SEMI_APDS_VAL;
    }

    else if (idc_values [HULL_HEAT])
    {
        controls_transfer_hull_heat ();
        ammo_transfer_hull_heat ();
        ammo_transfer_status = HULL_HEAT_VAL;
    }

    else if (idc_values [HULL_APDS])
    {
        controls_transfer_hull_apds ();
        ammo_transfer_hull_apds ();
        ammo_transfer_status = HULL_APDS_VAL;
    }

    else if (idc_values [REDIST_SEND])
    {
        controls_transfer_redist_send ();
        ammo_transfer_redist_send ();
        ammo_transfer_status = REDIST_SEND_VAL;
    }

    else if (idc_values [REDIST_RECV])
    {
        controls_transfer_redist_recv ();
        ammo_transfer_redist_recv ();
        ammo_transfer_status = REDIST_RECV_VAL;
    }

    else
    {
        controls_transfer_no_transfer ();
        ammo_transfer_no_transfer ();
        ammo_transfer_status = NO_TRANSFER_VAL;
    }
}


static void controls_transfer_semi_heat ()
{
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_SEND_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_RECV_L, OUTPUT_OFF);
    if ((ammo_get_quantity (SEMI_HEAT_VAL) > 0) &&
        (idc_values [SEMI_HEAT_L] != ON))
    {
        idc_output_set_cond  (! controls_failure_status (),
                           SEMI_HEAT_L, OUTPUT_ON);
    }
}


static void controls_transfer_semi_apds ()
{
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_SEND_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_RECV_L, OUTPUT_OFF);
    if ((ammo_get_quantity (SEMI_APDS_VAL) > 0) &&
        (idc_values [SEMI_APDS_L] != ON))
    {
        idc_output_set_cond  (! controls_failure_status (),
                           SEMI_APDS_L, OUTPUT_ON);
    }
}


static void controls_transfer_hull_heat ()
{
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_SEND_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_RECV_L, OUTPUT_OFF);
    if ((ammo_get_quantity (HULL_HEAT_VAL) > 0) &&
        (idc_values [HULL_HEAT_L] != ON))
    {
        idc_output_set_cond  (! controls_failure_status (),
                           HULL_HEAT_L, OUTPUT_ON);
    }
}


static void controls_transfer_hull_apds ()
{
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_SEND_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_RECV_L, OUTPUT_OFF);
    if ((ammo_get_quantity (HULL_APDS_VAL) > 0) &&
        (idc_values [HULL_APDS_L] != ON))
    {
        idc_output_set_cond  (! controls_failure_status (),
                           HULL_APDS_L, OUTPUT_ON);
    }
}


static void controls_transfer_no_transfer ()
{
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_SEND_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_RECV_L, OUTPUT_OFF);
}


static void controls_transfer_redist_send ()
{
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_RECV_L, OUTPUT_OFF);
    idc_output_set_cond  (! controls_failure_status (),
                       REDIST_SEND_L, OUTPUT_ON);
}


static void controls_transfer_redist_recv ()
{
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       HULL_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_HEAT_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       SEMI_APDS_L, OUTPUT_OFF);
    idc_output_set_cond (! controls_commander_panel_status (),
                       REDIST_SEND_L, OUTPUT_OFF);
    idc_output_set_cond  (! controls_failure_status (),
                       REDIST_RECV_L, OUTPUT_ON);
}


void controls_resupply_empty (status)
register int status;
{
    switch (status)
    {
        case HULL_HEAT_VAL:
            idc_output_set_cond (! controls_commander_panel_status (),
                                   HULL_HEAT_L, OUTPUT_OFF);
            break;

        case HULL_APDS_VAL:
            idc_output_set_cond (! controls_commander_panel_status (),
                                   HULL_APDS_L, OUTPUT_OFF);
            break;

        case SEMI_HEAT_VAL:
            idc_output_set_cond (! controls_commander_panel_status (),
                                   SEMI_HEAT_L, OUTPUT_OFF);
            break;

        case SEMI_APDS_VAL:
            idc_output_set_cond (! controls_commander_panel_status (),
                                   SEMI_APDS_L, OUTPUT_OFF);
            break;
        default:
            fprintf (stderr, "CONTROLS: controls_resupply_empty: Impossible status\
            nprintf ("CONTROLS: controls_resupply_empty: Impossible status\n");
            break;
    }
}


static void controls_ammo_tube_check ()
{
    register int        i;
    register int        retval;


    retval = NULL_SLOT;                             /* no tube selected */

    for (i = 0; i < N_READY; i++)                   /* find one selected */
    {
        if (idc_values [select_translations [i]])
        {
            retval = i;
            idc_values [select_translations [i]] = OFF;
        }
    }

    if (retval != NULL_SLOT)                        /* return selected - offset */
        ammo_tube_selected (retval);
}


void controls_show_round (slot, contents)
int slot;
ObjectType contents;
{
    if (slot == NULL_SLOT)
        return;

    if (contents == munition_US_M456A1)
        idc_output_set_cond (! controls_failure_status (),
                              heat_translations [slot], OUTPUT_ON);

    if (contents == munition_US_M392A2)
        idc_output_set_cond (! controls_failure_status (),
                              apds_translations [slot], OUTPUT_ON);
}
```

```
void controls_unshow_round (slot, contents)
int slot;
ObjectType contents;
{
    if (slot == NULL_SLOT)
        return;

    if (contents == munition_US_M456A1)
        idc_output_set_cond (! controls_commander_panel_status (),
                           heat_translations [slot], OUTPUT_OFF);

    if (contents == munition_US_M392A2)
        idc_output_set_cond (! controls_commander_panel_status (),
                           apds_translations [slot], OUTPUT_OFF);
}


static void controls_commander_weapon_station_check ()
{
    int temp;

    if((temp = ain(TC_CH)) == -1)
    {
        Dtad_Failed();
        return;
    }

    if (!using_polhemus) {              /* see polhemus note above */
        if ((temp > hex_comm_weap_val + DTAD_HYSTERESIS) ||
            (temp < hex_comm_weap_val - DTAD_HYSTERESIS))
        {
            real_comm_weap_val = pots_comm_weap_real (hex_comm_weap_val = temp);
            cupola_cws_new_value (real_comm_weap_val);
        }
    }
}


static void controls_commander_weapon_station_init ()
{
    if((hex_comm_weap_val = ain(TC_CH)) == -1)
    {
        hex_comm_weap_val = 0;
        Dtad_Failed();
        return;
    }

    real_comm_weap_val = pots_comm_weap_real (hex_comm_weap_val);
    cupola_cws_new_value (real_comm_weap_val);
}

static void controls_loader_periscope_check ()
{
    int temp;

    if((temp = ain(LD_CH)) == -1)
    {
        Dtad_Failed();
        return;
    }
    if ((temp > hex_load_peri_val + DTAD_HYSTERESIS) ||
        (temp < hex_load_peri_val - DTAD_HYSTERESIS))
    {
        real_load_peri_val = pots_load_peri_real (hex_load_peri_val = temp);
        cupola_lpscope_new_value (real_load_peri_val);
    }
}


static void controls_loader_periscope_init ()
{
    if((hex_load_peri_val = ain(LD_CH)) == -1)
    {
        hex_load_peri_val = 0;
        Dtad_Failed();
        return;
    }

    real_load_peri_val = pots_load_peri_real (hex_load_peri_val);
    cupola_lpscope_new_value (real_load_peri_val);
}


void controls_resupply_flash (slot, transfer_status, resupply_location)
int slot;
char transfer_status, resupply_location;
{
    switch (transfer_status)
    {
        case HULL_HEAT_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_set_ns (heat_translations [slot], OUTPUT_ON);
            }
            idc_output_set_ns (HULL_HEAT_L, OUTPUT_ON);
            break;
        case HULL_APDS_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_set_ns (apds_translations [slot], OUTPUT_ON);
            }
            idc_output_set_ns (HULL_APDS_L, OUTPUT_ON);
            break;
        case SEMI_HEAT_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_set_ns (heat_translations [slot], OUTPUT_ON);
            }
            idc_output_set_ns (SEMI_HEAT_L, OUTPUT_ON);
            break;
        case SEMI_APDS_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_set_ns (apds_translations [slot], OUTPUT_ON);
            }
            idc_output_set_ns (SEMI_APDS_L, OUTPUT_ON);
            break;
        case REDIST_SEND_VAL:
            idc_output_set_ns (REDIST_SEND_L, OUTPUT_ON);
            break;
        case REDIST_RECV_VAL:
            switch (resupply_location)
            {
                case HULL_HEAT_VAL:
                    idc_output_set_ns (HULL_HEAT_L, OUTPUT_ON);
                    idc_output_set_ns (REDIST_RECV_L, OUTPUT_ON);
                    break;
                case HULL_APDS_VAL:
                    idc_output_set_ns (HULL_APDS_L, OUTPUT_ON);
                    idc_output_set_ns (REDIST_RECV_L, OUTPUT_ON);
                    break;
                case SEMI_HEAT_VAL:
                    idc_output_set_ns (SEMI_HEAT_L, OUTPUT_ON);
                    idc_output_set_ns (REDIST_RECV_L, OUTPUT_ON);
                    break;
                case SEMI_APDS_VAL:
                    idc_output_set_ns (SEMI_APDS_L, OUTPUT_ON);
                    idc_output_set_ns (REDIST_RECV_L, OUTPUT_ON);
                    break;
                case READY_HEAT_VAL:
                    if (slot != NULL_SLOT)
                    {
                        idc_output_set_ns (heat_translations [slot], OUTPUT_ON);
                    }
                    idc_output_set_ns (REDIST_RECV_L, OUTPUT_ON);
                    break;
                case READY_APDS_VAL:
                    if (slot != NULL_SLOT)
                    {
                        idc_output_set_ns (apds_translations [slot], OUTPUT_ON);
                    }
                    idc_output_set_ns (REDIST_RECV_L, OUTPUT_ON);
                    break;
                default:
                    fprintf (stderr, "CONTROLS: controls_resupply_flash: Impossible
                    nprintf ("CONTROLS: controls_resupply_flash: Impossible resuppl
                    break;
            }
            break;
        default:
            fprintf (stderr, "CONTROLS: controls_resupply_flash: Impossible transfe
            nprintf ("CONTROLS: controls_resupply_flash: Impossible transfer_status
            break;
    }
}


void controls_resupply_unflash (slot, transfer_status, resupply_location)
int slot;
char transfer_status, resupply_location;
{
    switch (transfer_status)
    {
        case HULL_HEAT_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      heat_translations [slot], OUTPUT_OFF);
            }
            idc_output_set_ns_cond (! controls_commander_panel_status (),
                                  HULL_HEAT_L, OUTPUT_OFF);
            break;
        case HULL_APDS_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      apds_translations [slot], OUTPUT_OFF);
            }
            idc_output_set_ns_cond (! controls_commander_panel_status (),
                              HULL_APDS_L, OUTPUT_OFF);
        break;
    case SEMI_HEAT_VAL:
        if (slot != NULL_SLOT)
        {
            idc_output_set_ns_cond (! controls_commander_panel_status (),
                                  heat_translations [slot], OUTPUT_OFF);
        }
        idc_output_set_ns_cond (! controls_commander_panel_status (),
                              SEMI_HEAT_L, OUTPUT_OFF);
        break;
    case SEMI_APDS_VAL:
        if (slot != NULL_SLOT)
        {
            idc_output_set_ns_cond (! controls_commander_panel_status (),
                                  apds_translations [slot], OUTPUT_OFF);
        }
        idc_output_set_ns_cond (! controls_commander_panel_status (),
                              SEMI_APDS_L, OUTPUT_OFF);
        break;
    case REDIST_SEND_VAL:
        idc_output_set_ns_cond (! controls_commander_panel_status (),
                                  REDIST_SEND_L, OUTPUT_OFF);
        break;
    case REDIST_RECV_VAL:
        switch (resupply_location)
        {
            case HULL_HEAT_VAL:
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      HULL_HEAT_L, OUTPUT_OFF);
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      REDIST_RECV_L, OUTPUT_OFF);
                break;
            case HULL_APDS_VAL:
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      HULL_APDS_L, OUTPUT_OFF);
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      REDIST_RECV_L, OUTPUT_OFF);
                break;
            case SEMI_HEAT_VAL:
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      SEMI_HEAT_L, OUTPUT_OFF);
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      REDIST_RECV_L, OUTPUT_OFF);
                break;
            case SEMI_APDS_VAL:
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      SEMI_APDS_L, OUTPUT_OFF);
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      REDIST_RECV_L, OUTPUT_OFF);
                break;
            case READY_HEAT_VAL:
                if (slot != NULL_SLOT)
                {
                    idc_output_set_ns_cond (! controls_commander_panel_status (
                                          heat_translations [slot], OUTPUT_OFF)
                }
                idc_output_set_ns_cond (! controls_commander_panel_status (),
                                      REDIST_RECV_L, OUTPUT_OFF);
                break;
            case READY_APDS_VAL:
                if (slot != NULL_SLOT)
                {
                    idc_output_set_ns_cond (! controls_commander_panel_status (
                                          apds_translations [slot], OUTPUT_OFF)
                }
```

```
                    idc_output_set_ns_cond (! controls_commander_panel_status (),
                                          REDIST_RECV_L, OUTPUT_OFF);
                    break;
                default:
                    fprintf (stderr, "CONTROLS: controls_resupply_unflash: Impossib
                    nprintf ("CONTROLS: controls_resupply_unflash: Impossible resup
                    break;
            }
            break;
        default:
            fprintf (stderr, "CONTROLS: controls_resupply_unflash: Impossible trans
            nprintf ("CONTROLS: controls_resupply_unflash: Impossible transfer_stat
            break;
    }
}


void controls_resupply_restore (slot, transfer_status, resupply_location)
int slot;
char transfer_status, resupply_location;
{
    switch (transfer_status)
    {
        case HULL_HEAT_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                       (! controls_failure_status ()),
                                       heat_translations [slot]);
            }
            idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                   (! controls_failure_status ()),
                                   HULL_HEAT_L);
            break;
        case HULL_APDS_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                       (! controls_failure_status ()),
                                       apds_translations [slot]);
            }
            idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                   (! controls_failure_status ()),
                                   HULL_APDS_L);
            break;
        case SEMI_HEAT_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                       (! controls_failure_status ()),
                                       heat_translations [slot]);
            }
            idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                   (! controls_failure_status ()),
                                   SEMI_HEAT_L);
            break;
        case SEMI_APDS_VAL:
            if (slot != NULL_SLOT)
            {
                idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                       (! controls_failure_status ()),
                                       apds_translations [slot]);
            }
            idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                   (! controls_failure_status ()),
                                   SEMI_APDS_L);
        break;
    case REDIST_SEND_VAL:
        idc_output_restore_cond ((! controls_commander_panel_status ()) &&
                                 (! controls_failure_status ()),
                                 REDIST_SEND_L);
        break;
    case REDIST_RECV_VAL:
        switch (resupply_location)
        {
            case HULL_HEAT_VAL:
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         HULL_HEAT_L);
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         REDIST_RECV_L);
                break;
            case HULL_APDS_VAL:
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         HULL_APDS_L);
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         REDIST_RECV_L);
                break;
            case SEMI_HEAT_VAL:
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         SEMI_HEAT_L);
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         REDIST_RECV_L);
                break;
            case SEMI_APDS_VAL:
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         SEMI_APDS_L);
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         REDIST_RECV_L);
                break;
            case READY_HEAT_VAL:
                if (slot != NULL_SLOT)
                {
                    idc_output_restore_cond ((! controls_commander_panel_status
                                             (! controls_failure_status ()),
                                             heat_translations [slot]);
                }
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         REDIST_RECV_L);
                break;
            case READY_APDS_VAL:
                if (slot != NULL_SLOT)
                {
                    idc_output_restore_cond ((! controls_commander_panel_status
                                             (! controls_failure_status ()),
                                             apds_translations [slot]);
                }
                idc_output_restore_cond ((! controls_commander_panel_status ())
                                         (! controls_failure_status ()),
                                         REDIST_RECV_L);
                break;
            default:
                fprintf (stderr, "CONTROLS: controls_resupply_restore: Impossib
                nprintf ("CONTROLS: controls_resupply_restore: Impossible resup
                        break;
                }
                break;
            case NO_TRANSFER_VAL:
                break;
            default:
                fprintf (stderr, "CONTROLS: controls_resupply_restore: Impossible trans
                nprintf ("CONTROLS: controls_resupply_restore: Impossible transfer_stat
                break;
        }
}


void controls_odometer_pulse ()
{
    idc_output_set (DR_ODOMETER_PULSE, OUTPUT_ON);

    /* If the odometer_timer is already timing, we must replace it */
    /* with the more up-to-date timer, and the old one must be freed */
    if (odometer_timer_number != NULL_TIMER)
    {
        timers_free_timer (odometer_timer_number);
    }

    odometer_timer_number = timers_get_timer (ODOMETER_DELAY);
}


void controls_turret_ref_ind (radians)
REAL radians;
{
    REAL degrees, shift_degrees;
    int offset, i;

    degrees = rad_to_deg (radians);

    if ((shift_degrees = degrees - (TURRET_REF_SECTOR_SIZE / 2.0)) < 0.0)
    {
        shift_degrees += 360.0;
    }

    offset = shift_degrees / TURRET_REF_SECTOR_SIZE;

    if (! idc_values [turret_ref_translations[offset]])
    {
        for (i = 0; i < TURRET_REF_NUM_SECTORS; i++)
        {
            if (i == offset)
            {
                idc_output_set_cond  (! controls_failure_status (),
                                   turret_ref_translations[i], OUTPUT_ON);
            }
            else if (idc_values [turret_ref_translations[i]])
            {
                idc_output_set_cond (! controls_commander_panel_status (),
                                   turret_ref_translations[i], OUTPUT_OFF);
            }
        }
    }
}


static void controls_grid_azimuth_check ()
{
    REAL speed;
    /* so abs doesn't call tracks_compute_velocity () twice ... */
    speed = tracks_compute_velocity ();

    /* need to check this every tick in case you start moving
       and the azimuth button is pushed, the azimuth indicator
       should turn off */
    switch (idc_values[GRID_AZIMUTH_PB])
    {
        case ON:
            map_set_bumper_status (TRUE);
            if (abs(speed) < 0.1)
            {
                if (! grid_azimuth_status)
                {
                    turret_send_azimuth_ind ();
                    grid_azimuth_status = ON;
                }
            }
            else
            {
                if (grid_azimuth_status)
                {
                    turret_null_azimuth_ind ();
                    grid_azimuth_status = OFF;
                }
            }
            break;
        case OFF:
            map_set_bumper_status (FALSE);
            if (grid_azimuth_status)
            {
                turret_null_azimuth_ind ();
                grid_azimuth_status = OFF;
            }
            break;
        default:
            fprintf (stderr, "CONTROLS: controls_grid_azimuth_check: Impossible pus
            nprintf ("CONTROLS: controls_grid_azimuth_check: Impossible push button
            break;
    }
}


void controls_show_breech (contents)
ObjectType contents;
{
    if ((contents == munition_US_M456A1) ||
        (contents == munition_US_M392A2))
    {
        idc_output_set_cond (! controls_failure_status (),
                             LD_BREECH_LOADED_L, OUTPUT_ON);
    }

    else
    {
        idc_output_set_cond (! controls_commander_panel_status (),
                             LD_BREECH_LOADED_L, OUTPUT_OFF);
    }
}


void controls_no_power_off ()
{
    idc_reset ();
    timers_delay_proc (INVERT_DELAY, idc_invert_outputs, NECESSARY, 0.0);
    controls_service_brake_exit ();
    ammo_stop_timers ();
    controls_odometer_exit ();
    controls_cupola_up_down_exit ();
    controls_binoculars_on_off_exit ();
    controls_lpscope_up_down_exit ();
}


static void controls_fuel_flash_check ()
{
    if ((fuel_flash_status == ON) &&
        (++fuel_flash_count == TICKS_PER_SECOND))
    {
        fuel_flash_count = 0;
    }

    if (fuel_flash_count == BEGIN_FLASH)
    {
        controls_fuel_flash ();
    }

    else if (fuel_flash_count == END_FLASH)
    {
        controls_fuel_unflash ();
    }
}


void controls_start_fuel_flashing ()
{
    fuel_flash_status = ON;
    fuel_flash_count = 0;
    controls_fuel_restore ();
}


void controls_stop_fuel_flashing ()
{
    fuel_flash_status = OFF;
    fuel_flash_count = 0;
    controls_fuel_restore ();
}


static void controls_fuel_flash ()
{
    idc_output_set_ns (DR_LOW_FUEL_L, OUTPUT_ON);
}


static void controls_fuel_unflash ()
{
    idc_output_set_ns_cond (! controls_driver_panel_status (),
                            DR_LOW_FUEL_L, OUTPUT_OFF);
}


static void controls_fuel_restore ()
{
    idc_output_restore_cond ((! controls_driver_panel_status ()) &&
                             (! controls_failure_status ()),
                             DR_LOW_FUEL_L);
}
```

```c
static void controls_odometer_check ()
{
    if (timers_get_timeout_edge (odometer_timer_number))
    {
        timers_free_timer (odometer_timer_number);
        odometer_timer_number = timers_set_null_timer ();
        idc_output_set (DR_ODOMETER_PULSE, OUTPUT_OFF);
    }
}



static void controls_odometer_exit ()
{
    timers_free_timer (odometer_timer_number);
    odometer_timer_number = timers_set_null_timer ();
}


static void controls_cupola_up_down_check ()
{
    char temp_up, temp_down;

    temp_up = idc_values [CM_CUPOLA_UP];
    temp_down = idc_values [CM_CUPOLA_DOWN];

    if ((temp_up) &&
        (cupola_up_down_val != CM_UP_VAL) &&
        (! temp_down))
    {
        cupola_up_down_val = CM_UP_VAL;
/*      vision_cmdrs_pitch_up ();*/
        vision_cmdrs_pitch (PITCH_UP);
    }

    else if ((! temp_up) &&
        (cupola_up_down_val != CM_CENTER_VAL) &&
        (! temp_down))
    {
        cupola_up_down_val = CM_CENTER_VAL;
/*      vision_cmdrs_pitch_ahead ();*/
        vision_cmdrs_pitch (PITCH_AHEAD);
    }

    else if ((temp_down) &&
        (cupola_up_down_val != CM_DOWN_VAL) &&
        (! temp_up))
    {
        cupola_up_down_val = CM_DOWN_VAL;
/*      vision_cmdrs_pitch_down ();*/
        vision_cmdrs_pitch (PITCH_DOWN);
    }
}


static void controls_cupola_up_down_init ()
{
    if (idc_values [CM_CUPOLA_UP])
    {
        cupola_up_down_val = CM_UP_VAL;
/*      vision_cmdrs_pitch_up ();*/
        vision_cmdrs_pitch (PITCH_UP);
    }

    else if (idc_values [CM_CUPOLA_DOWN])
    {
        cupola_up_down_val = CM_DOWN_VAL;
/*      vision_cmdrs_pitch_down ();*/
        vision_cmdrs_pitch (PITCH_DOWN);
    }

    else
    {
        cupola_up_down_val = CM_CENTER_VAL;
/*      vision_cmdrs_pitch_ahead ();*/
        vision_cmdrs_pitch (PITCH_AHEAD);
    }
}


static void controls_cupola_up_down_exit ()
{
/*    vision_cmdrs_pitch_ahead ();*/
    vision_cmdrs_pitch (PITCH_AHEAD);
}


static void controls_lpscope_up_down_check ()
{
    char temp_up, temp_down;

    temp_up = idc_values [LD_PSCOPE_UP];
    temp_down = idc_values [LD_PSCOPE_DOWN];

    if ((temp_up) &&
        (lpscope_up_down_val != LD_UP_VAL) &&
        (! temp_down))
    {
        lpscope_up_down_val = LD_UP_VAL;
/*      vision_loaders_pitch_up ();*/
        vision_loaders_pitch (PITCH_UP);
    }

    else if ((! temp_up) &&
        (lpscope_up_down_val != LD_CENTER_VAL) &&
        (! temp_down))
    {
        lpscope_up_down_val = LD_CENTER_VAL;
/*      vision_loaders_pitch_ahead ();*/
        vision_loaders_pitch (PITCH_AHEAD);
    }

    else if ((temp_down) &&
        (lpscope_up_down_val != LD_DOWN_VAL) &&
        (! temp_up))
    {
        lpscope_up_down_val = LD_DOWN_VAL;
/*      vision_loaders_pitch_down ();*/
        vision_loaders_pitch (PITCH_DOWN);
    }
}


static void controls_lpscope_up_down_init ()
{
    if (idc_values [LD_PSCOPE_UP])
    {
        lpscope_up_down_val = LD_UP_VAL;
/*      vision_loaders_pitch_up ();*/
        vision_loaders_pitch (PITCH_UP);
    }
    else if (idc_values [LD_PSCOPE_DOWN])
    {
        lpscope_up_down_val = LD_DOWN_VAL;
/*      vision_loaders_pitch_down ();*/
        vision_loaders_pitch (PITCH_DOWN);
    }

    else
    {
        lpscope_up_down_val = LD_CENTER_VAL;
/*      vision_loaders_pitch_ahead ();*/
        vision_loaders_pitch (PITCH_AHEAD);
    }
}


static void controls_lpscope_up_down_exit ()
{
/*    vision_loaders_pitch_ahead ();*/
    vision_loaders_pitch (PITCH_AHEAD);
}


void controls_restore_ammo ()
{
    switch (ammo_transfer_status)
    {
        case SEMI_HEAT_VAL:
            controls_transfer_semi_heat ();
            break;
        case SEMI_APDS_VAL:
            controls_transfer_semi_apds ();
            break;
        case HULL_HEAT_VAL:
            controls_transfer_hull_heat ();
            break;
        case HULL_APDS_VAL:
            controls_transfer_hull_apds ();
            break;
        case NO_TRANSFER_VAL:
            controls_transfer_no_transfer ();
            break;
        case REDIST_SEND_VAL:
            controls_transfer_redist_send ();
            break;
        case REDIST_RECV_VAL:
            controls_transfer_redist_recv ();
            break;
        default:
            fprintf (stderr, "CONTROLS: controls_restore_ammo: Impossible ammo tran
            nprintf ("CONTROLS: controls_restore_ammo: Impossible ammo transfer sta
            break;
    }
}

static void
controls_binoculars_on_off_exit ()
{
    binoculars_on_off_val = OFF;
}

static void
controls_binoculars_on_off_init ()
{
    binoculars_on_off_val = idc_values[CM_BINOCULARS];
    if (binoculars_on_off_val == ON)
```

```
        vision_cmdrs_binoculars (BINOC);
    else
        vision_cmdrs_binoculars (NO_BINOC);
}

static void
controls_binoculars_on_off_check ()
{
char temp;

    if ((temp = idc_values[CM_BINOCULARS]) != binoculars_on_off_val)
        {
        binoculars_on_off_val = temp;
        if (binoculars_on_off_val == ON)
            vision_cmdrs_binoculars (BINOC);
        else
            vision_cmdrs_binoculars (NO_BINOC);
        }

}


static void
controls_thermal_shutter_check ()
{
    char temp;

    if ((temp = idc_values [THERM_SHUTTER]) != thermal_shutter_val)
    {
        switch (thermal_shutter_val = temp)
        {
            case ON:
                thermal_shutter();
                break;
            case OFF:
                thermal_clear();
                break;
            default:
                fprintf (stderr, "CONTROLS: controls_thermal_shutter_check: Impossi
                printf ("CONTROLS: controls_thermal_shutter_check: Impossible therm
                break;
        }
    }
}


static void
controls_thermal_shutter_init ()
{
    vision_set_gunner_no_thermal();
    switch (thermal_shutter_val = idc_values [THERM_SHUTTER])
    {
        case ON:
            thermal_shutter();
            thermal_shutter_val = OFF;
            break;
        case OFF:
            thermal_clear();
            thermal_shutter_val = ON;
            break;
        default:
            fprintf (stderr, "CONTROLS: controls_thermal_shutter_check: Impossible
            printf ("CONTROLS: controls_thermal_shutter_check: Impossible thermal_s
            break;
    }
}
```

```
/*********************************************************************
 *                                                                   *
 * FILE:         ml_keybrd.c                                         *
 * AUTHOR:       Brian O'Toole                                       *
 * MAINTAINER:   Brian O'Toole                                       *
 * HISTORY:      4/30/86  brian: Creation                            *
 *                                                                   *
 *                                                                   *
 * Copyright (c) 1986 BBN Laboratories, Inc.                         *
 * All rights reserved.                                              *
 *                                                                   *
 *********************************************************************/

#include "stdio.h"
#include "fcntl.h"

#ifndef SIMBFLY

#include "signal.h"
#include "termio.h"

#endif

#include "sim_dfns.h"
#include "sim_types.h"
#include "sim_macros.h"

#include "rtc.h"

#include "pro_sim.h"
#include "pro_data.h"
#include "repair_ml.h"

#include "mass_stdc.h"
#include "dgi_stdg.h"
#include "sim_cig_if.h"

#include "libfail.h"
#include "librepair.h"
#include "libmem_dfn.h"
#include "libnetwork.h"
#include "timers.h"
#include "libsound.h"
#include "libhull.h"
#include "libkin.h"
#include "libfilter.h"
#include "librva.h"

#include "ml_main.h"
#include "ml_cntrl.h"
#include "ml_elecsys.h"
#include "ml_engine.h"
#include "ml_laser.h"
#include "ml_fuelsys.h"
#include "ml_weapons.h"
#include "ml_tracks.h"
#include "ml_turret.h"
#include "ml_dtrain.h"
#include "ml_vision.h"
#include "ml_repair.h"
#include "ml_sound.h"
#include "ml_ammo.h"
#include "ml_keybrd.h"
#include "ml_polhemus.h"
#include "ml_cupola.h"
#include "status.h"
```

```c
#include "m1_status.h"
#include "m1_turr_pn.h"
#include "m1_driv_pn.h"
#include "net/network.h"

#ifndef SIMBFLY

#include "enpioctl.h"

#else

#include "enpsvr.h"

#endif

#define DELTA_POT 0.005

static int console;
static int use_keyboard = FALSE;
static int use_cupola = FALSE;
static REAL lpscope_value = 0.0;
static REAL cws_value = 0.0;
static REAL* vec;

static void keyboard_setup_terminal ();

void keyboard_really_use ()
{
    use_keyboard = TRUE;
}


void keyboard_use_cupola ()
{
    use_cupola = TRUE;
    printf ("Cupola and periscope now under keyboard control\n");
}


void keyboard_init ()
{
    if (! use_keyboard)
    {
        return;
    }

    keyboard_setup_terminal ();
    printf ("Keyboard ready: type <?> for help\n");

    if (use_cupola)
    {
        cupola_lpscope_new_value (0.0);
        cupola_cws_new_value (0.0);
    }
}

#ifdef SIMBFLY
/* Want to know how many pkts/second causes simulation to degrade */
static long pkt_cnt_start;
#endif

void keyboard_simul ()
{
    char cmd;
    int network_stats[N_STATS];
    char network_statstr[41];
    int n;

    if (! use_keyboard)
    {
        return;
    }

    cmd = keybrd_tty_read (console);
    if (cmd == 0)
        return;

    switch (cmd & 0x7F)                 /* want 7 bit ascii */
    {
        case 'a' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_CommAntennaFailure);*/
            printf ("keyboard: controls_kill_radio ()\n");
            break;
        case 'A' :
            reconstitute_from_keyboard ();/* changed by cjc 2/14/89 */
                                                                          /* change

                 printf ("keyboard: reconstitute_vehicle ()\n");
/*
            repair_stop_repair(M1_CommAntennaFailure);
            printf ("keyboard: controls_restore_radio ()\n");
*/
            break;
        case 'b' :
            polhemus_init();
            set_polhemus_flag_true();
            controls_set_using_polhemus_true();
            printf ("keyboard: Use Polhemus for cupola\n");
            break;
        case 'B' :
            set_polhemus_flag_false();
            controls_set_using_polhemus_false();
            polhemus_exit();
            printf ("keyboard: Turn off Polhemus.\n");
            break;
        case 'c' :
            controls_break_controls ();
            printf ("keyboard: controls_break_controls ()\n");
            break;
        case 'C' :
            controls_restore_controls ();
            printf ("keyboard: controls_restore_controls ()\n");
            break;
        case 'd' :
            printf("keyboard: before deactivate simulation\n");
            deactivate_simulation ();
            printf("keyboard: after deactivate simulation\n");
            break;
        case 'D':
            filter_dump_filter_info();
            break;
        case 'e' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_EngineMajorFailure );*/
            printf ("keyboard: engine_major_failure ()\n");
            break;
        case 'E' :
            repair_fix_system( repairCauseIntervention, m1ReplacePowerPack );
            printf ("keyboard: engine_replace_powerpack ()\n");
            break;
        case 'f' :
            printf("keyboard: network_get_vehicle_force() = %d\n",
                        network_get_vehicle_force() );
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_LRFFailure );
            printf ("keyboard: laser_lrf_failure ()\n"); */
            break;
        case 'F' :
            repair_fix_system( repairCauseIntervention, m1RepairLRF );
            printf ("keyboard: laser_repair_lrf ()\n");
            break;
        case 'g' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_EngineOilFilterClogged );*/
/*          printf ("keyboard: engine_clog_oil_filter ()\n");*/
            printf ("turning on asid_debug\n");
            map_set_asid_debug (TRUE);
            break;
        case 'G' :
/*          repair_fix_system( repairCauseIntervention, m1ReplaceEngineOilFilter );
/*          printf ("keyboard: engine_replace_oil_filter ()\n");*/
            printf ("turning off asid_debug\n");
            map_set_asid_debug (FALSE);
            break;
        case 'h' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_FuelTransferPumpFailure );*/
            printf ("keyboard: fuel_transfer_pump_failure ()\n");
            break;
        case 'H' :
            repair_fix_system( repairCauseIntervention, m1RepairFuelTransferPump );
            printf ("keyboard: fuel_repair_transfer_pump ()\n");
            break;
        case 'i' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_TurretMainGunFailure );*/
            printf ("keyboard: weapons_disable_main_gun ()\n");
            break;
        case 'I' :
            repair_fix_system( repairCauseIntervention, m1RepairTurretMountInterfac
            printf ("keyboard: weapons_repair_main_gun ()\n");
            break;
        case 'j' :
            electsys_battery_failure ();
            printf ("keyboard: electsys_battery_failure ()\n");
            break;
        case 'J' :
            repair_fix_system( repairCauseIntervention, m1ReplaceBattery );
            printf ("keyboard: electsys_replace_battery ()\n");
            break;
        case 'k' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_EngineStarterFailure );*/
            printf ("keyboard: engine_starter_failure ()\n");
            break;
        case 'K' :
            repair_fix_system( repairCauseIntervention, m1ReplacePilotRelayStarter
            printf ("keyboard: engine_replace_starter ()\n");
            break;
        case 'l' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_FDriveLeftTrackFailure );*/
            printf ("keyboard: tracks_throw_left_track ()\n");
            break;
        case 'L' :
            printf("keyboard: rva_dump_priority_lists()\n");
            rva_dump_priority_lists();
            break;
        case 'm' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_TurretGunMountFailure );
            printf ("keyboard: turret_break_mount_interface ()\n");*/
            map_print ();
            break;
        case 'M' :
            repair_fix_system( repairCauseIntervention, mlRepairTurretMountInterfac
            printf ("keyboard: turret_repair_mount_interface ()\n");
            break;
        case 'n' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_TurretGunElevationFailure );*/
            printf ("keyboard: turret_break_elevation_drive ()\n");
            break;
        case 'N' :
            repair_fix_system( repairCauseIntervention, mlRepairGunElevationDrive )
            printf ("keyboard: turret_repair_elevation_drive ()\n");
            break;
        case 'o' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_DTrainOilFilterClogged );*/
            printf ("keyboard: drivetrain_clog_transmission_oil_filter ()\n");
            break;
        case 'O' :
            repair_fix_system( repairCauseIntervention, mlReplaceTransOilFilter );
            printf("keyboard: drivetrain_replace_transmission_oil_filter ()\n");
            break;
        case 'p' :
            rva_turn_debug_on();
            printf("Turning priority sort debug on\n");
            break;
        case 'P' :
            rva_turn_debug_off();
            printf("Turning priority sort debug off\n");
            break;
        case 'q' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_CmdrsVisionBlocksBroken );*/
            printf ("keyboard: vision_break_cmdrs_blocks ()\n");
            break;
        case 'Q' :
/*          repair_stop_repair( M1_CmdrsVisionBlocksBroken );*/
            printf ("keyboard: vision_restore_cmdrs_blocks ()\n");
            break;
        case 'r' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_CommAntennaFailure );*/
            controls_kill_radio();
            printf ("keyboard: controls_kill_radio ()\n");
            break;
        case 'R' :
/*          repair_stop_repair( M1_CommAntennaFailure );*/
            controls_restore_radio();
            printf ("keyboard: controls_restore_radio ()\n");
            break;
        case 's' :
/*          fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                                M1_TurretStabSystemFailure );
            printf ("keyboard: turret_break_stab_system ()\n");*/
            use_static_debug (1);
            break;
        case 'S' :
/*          repair_fix_system( repairCauseIntervention, mlRepairStabSystem );
            printf ("keyboard: turret_repair_stab_system ()\n");
```

```
    case '1' :
/*      fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                           M1_DriversVisionBlocksBroken );*/
        printf ("keyboard: vision_break_driver_blocks ()\n");
        break;
    case '!' :
/*      repair_stop_repair( M1_DriversVisionBlocksBroken );*/
        printf ("keyboard: vision_restore_driver_blocks ()\n");
        break;
    case '2' :
/*      fail_break_system( vehicleIDIrrelevant, damageCauseIntervention,
                           M1_GunnersSightBroken );*/
        printf ("keyboard: vision_break_gps ()\n");
        break;
    case '@' :
/*      repair_stop_repair( M1_GunnersSightBroken );*/
        printf ("keyboard: vision_restore_gps ()\n");
        break;
    case '3' :
        controls_electsys_dead ();
        printf ("keyboard: controls_electsys_dead ()\n");
        break;
    case '#' :
        controls_electsys_reborn ();
        printf ("keyboard: controls_electsys_reborn ()\n");
        break;
    case '4' :
        sound_reset ();
        printf ("keyboard: sound_reset ()\n");
        break;
    case '5' :
        fail_cat_kill ( &vehicleIDIrrelevant, damageCauseIntervention );
        printf ("keyboard: fail_cat_kill ()\n");
        break;
    case '=' :
        fuel_init_tanks (187.0, 70.0, 70.0);
        printf ("keyboard: fuel_init_tanks ()\n");
        break;
    case '%' :
        repair_all_systems ();
        controls_electsys_reborn ();
        controls_restore_radio ();
        if (use_cupola)
        {
            cupola_lpscope_new_value (lpscope_value);
            cupola_cws_new_value (cws_value);
        }
        printf ("keyboard: fixing everything\n");
        break;
    case '[' :
        printf ("keyboard: network_print_statistics ()\n");
        network_print_statistics ();
        break;
    case ']' :
        printf ("keyboard: temperature and power supplies\n");
        status_print_temp_and_supplies ();
        break;
    case '^' :
        if (use_cupola)
        {
            printf ("keyboard: loader's periscope left\n");
            lpscope_value -= DELTA_POT;
            if (lpscope_value < -1.0)
                lpscope_value = -1.0;
            cupola_lpscope_new_value (lpscope_value);
        }
```

```
            else
            {
                printf ("toggling gunners vision\n");
                toggle_gunner_vision_state ();
            }
            break;
        case '&' :
            if (use_cupola)
            {
                printf ("keyboard: loader's periscope right\n");
                lpscope_value += DELTA_POT;
                if (lpscope_value > 1.0)
                    lpscope_value = 1.0;
                cupola_lpscope_new_value (lpscope_value);
            }
            else
            {
                printf ("toggling drivers vision\n");
                toggle_driver_vision_state ();
            }
            break;
        case '*' :
            if (use_cupola)
            {
                printf ("keyboard: commander's cupola left\n");
                cws_value -= DELTA_POT;
                if (cws_value < -1.0)
                    cws_value = -1.0;
                cupola_cws_new_value (cws_value);
            }
            else
            {
                printf ("view modes\n");
                print_view_modes ();
            }
            break;
        case '(' :
            if (use_cupola)
            {
                printf ("keyboard: commander's cupola right\n");
                cws_value += DELTA_POT;
                if (cws_value > 1.0)
                    cws_value = 1.0;
                cupola_cws_new_value (cws_value);
            }
            else
            {

                printf ("Unassigned character: type <?> for help\n");
            }
            break;
        case '?' :
/* 1 */     HELP_PRINT1 ('A', "reconstitute_vehicle");
/* 2 */     HELP_PRINT2 ('b', "Use Polhemus for cupola.", 'B',
                "Turn off Polhemus.");
/* 3 */     HELP_PRINT2 ('c', "controls_break_controls", 'C',
                "controls_restore_controls");
/* 4 */     HELP_PRINT1 ('D', "filter_dump_filter_info");
/* 5 */     HELP_PRINT2 ('e', "engine_major_failure", 'E',
                "engine_replace_powerpack");
/* 6 */     HELP_PRINT2 ('f', "laser_lrf_failure", 'F', "laser_repair_lrf");
/* 7 */     HELP_PRINT2 ('g', "engine_clog_oil_filter", 'G',
                "engine_replace_oil_filter");
/* 8 */     HELP_PRINT2 ('h', "fuel_transfer_pump_failure", 'H',
                "fuel_repair_transfer_pump");
/* 9 */     HELP_PRINT2 ('i', "weapons_disable_main_gun", 'I',
                 "weapons_repair_main_gun");
/* 10 */    HELP_PRINT2 ('j', "electsys_battery_failure", 'J',
                 "electsys_replace_battery");
/* 11 */    HELP_PRINT2 ('k', "engine_starter_failure", 'K',
                 "engine_replace_starter");
/* 12 */    HELP_PRINT1 ('l', "tracks_throw_left_track");
/* 13 */    HELP_PRINT1 ('L', "rva_dump_priority_lists");
/* 14 */    HELP_PRINT2 ('m', "turret_break_mount_interface", 'M',
                 "turret_repair_mount_interface");
/* 15 */    HELP_PRINT2 ('n', "turret_break_elevation_drive", 'N',
                 "turret_repair_elevation_drive");
/* 16 */    HELP_PRINT2 ('o', "drivetrain_clog_transmission_oil_filter", 'O',
                 "drivetrain_replace_transmission_oil_filter");
/* 17 */    HELP_PRINT2 ('p', "rva_turn_debug_on", 'P',
                 "rva_turn_debug_off");
/* 18 */    HELP_PRINT2 ('q', "vision_break_cmdrs_blocks", 'Q',
                 "vision_restore_cmdrs_blocks");
/* 19 */    HELP_PRINT2 ('r', "controls_kill_radio", 'R',
                 "controls_restore_radio");
/* 20 */    printf ("TO SEE THE NEXT PAGE, TYPE <6> ...\n");
            break;
        case '6' :
/* 1 */     HELP_PRINT2 ('s', "static_debug on", 'S', "static_debug off");
/* 2 */     HELP_PRINT2 ('t', "drivetrain_transmission_failure", 'T',
                 "engine_replace_power_pack");
/* 3 */     HELP_PRINT2 ('u', "engine_clog_fuel_filter", 'U',
                 "engine_replace_fuel_filter");
/* 4 */     HELP_PRINT2 ('v', "turret_break_traverse_drive", 'V',
                 "turret_repair_traverse_drive");
/* 5 */     HELP_PRINT2 ('w', "drivetrain_transmission_oil_leak", 'W',
                 "engine_replace_powerpack");
/* 6 */     HELP_PRINT2 ('x', "vision_break_gps_ext", 'X',
                 "vision_restore_gps_ext");
/* 7 */     HELP_PRINT2 ('y', "vision_break_ldrs_pscope" , 'Y',
                 "vision_restore_ldrs_pscope");
/* 8 */     HELP_PRINT2 ('z', "engine_oil_leak", 'Z',
                 "engine_replace_powerpack");
/* 9 */     HELP_PRINT2 ('1', "vision_break_driver_blocks", '!',
                 "vision_restore_driver_blocks");
/* 10 */    HELP_PRINT2 ('2', "vision_break_gps", '@', "vision_restore_gps");
/* 11 */    HELP_PRINT2 ('3', "controls_electsys_dead", '#',
                 "controls_electsys_reborn");
/* 12 */    HELP_PRINT1 ('4', "sound_reset");
/* 13 */    HELP_PRINT2 ('5', "fail_cat_kill", '%', "fixing everything");
/* 14 */    HELP_PRINT1 ('=', " fuel_tanks_init");
/* 15 */    HELP_PRINT1 ('[', "network_print_statistics");
/* 16 */    HELP_PRINT1 (']', " print temperature and power supplies");
            if (use_cupola)
            {
/* 17 */        HELP_PRINT2 ('^', "loader's periscope left", '&',
                     "loader's periscope right");
/* 18 */        HELP_PRINT2 ('*', "commander's cupola left", '(',
                     "commander's cupola right");
            }
            else
            {
                HELP_PRINT2 ('^', "toggle gunners vision", '&',
                     "toggle drivers vision");
            }
/* 19 */    HELP_PRINT1 (',', "print_reasons");
/* 20 */    printf ("TO SEE THE NEXT PAGE, TYPE <7> ...\n");
            break;
        case '7' :
/* 1 */     HELP_PRINT2 ('0', "in pivot steer", ')', "out of pivot steer");
/* 2 */     HELP_PRINT2 ('/', "binoculars on", '\\', "binoculars off");
/* 3 */     HELP_PRINT1 ('9', "Restore ammo");
/* 4 */     HELP_PRINT1 ('-', "timers_status");
/* 5 */     HELP_PRINT1 ('+', "print CMC statistics");
/* 6 */     HELP_PRINT1 ('_', "zero CMC statistics");
/* 7 */     HELP_PRINT1 (';', "send_azimuth");
/* 8 */     HELP_PRINT1 (':', "null_azimuth");
/* 9 */     HELP_PRINT1 ('}', "print and reset bbd rtc statistics");
/* 10 */    HELP_PRINT1 ('{', "print bbd rtc statistics");
/* 11 */    HELP_PRINT1 ('<', "Current <x, y, z>");
/* 12 */    HELP_PRINT1 ('.', "ammo_print_statistics");
/* 13 */    HELP_PRINT1 ('~', "print n_mapped value");
/* 14 */    HELP_PRINT1 ('X', "print_sorted_vehicle_lists");
/* 15 */    HELP_PRINT1 ('|', "timing bits");
/* 16 */    HELP_PRINT1 ('?', "Page 1 of help");
/* 17 */    HELP_PRINT1 ('6', "Page 2 of help");
/* 18 */    HELP_PRINT1 ('7', "Page 3 of help");
            break;
        case '9' :
            ammo_restore_ammo ();
            controls_restore_ammo ();
            printf ("keyboard: ammo_restore_ammo ()\n");
            printf ("keyboard: controls_restore_ammo ()\n");
            break;
        case '0' :
            idc_values [DR_PIVOT_MODE] = 1;
            idc_values [DR_TRANS_NEUTRAL] = 0;
            idc_values [DR_TRANS_DRIVE] = 0;
            idc_values [DR_TRANS_LOW] = 0;
            idc_values [DR_TRANS_REVERSE] = 0;
            printf ("keyboard: in PIVOT steer\n");
            break;
        case ')' :
            idc_values [DR_PIVOT_MODE] = 0;
            idc_values [DR_TRANS_NEUTRAL] = 0;
            idc_values [DR_TRANS_DRIVE] = 0;
            idc_values [DR_TRANS_LOW] = 0;
            idc_values [DR_TRANS_REVERSE] = 0;
            printf ("keyboard: out of PIVOT steer\n");
            break;
        case '-' :
            printf ("keyboard: timers_status ()\n");
            timers_status ();
            break;
        case '+' :
            printf ("keyboard: net_getstats\n");
            if(net_print_statistics(net_handle) == -1)
            {
                printf("can't get network statistics\n");
                nprintf("KEYBOARD: can't get network statistics\n");
            }
#ifdef SIMBFLY
            else
            {
            long now;
            now = rtc;
            now -= pkt_cnt_start;
            printf("%6.3f pkts/second\n",
                    network_stats[12] * SECOND / ((double) now));
            }
#endif
            break;
        case '_' :
            printf ("keyboard: net_zerostats\n");
            if (net_zero_statistics(net_handle) == -1)
            {
                printf ("can't zero network statistics\n");
                nprintf ("KEYBOARD: can't zero network statistics\n");
            }
#ifdef SIMBFLY
            pkt_cnt_start = rtc;
#endif
            break;
        case ';' :
            printf ("keyboard: send_azimuth\n");
            turret_send_azimuth_ind();
            break;
        case ':' :
            printf ("keyboard: null_azimuth\n");
            turret_null_azimuth_ind();
            break;
        case '{':
#if defined(SIMBFLY)
            bbd_rtc_statistics(FALSE);
#else
            printf("{ is only available on the Butterfly\n");
#endif
            break;
        case '}':
#if defined(SIMBFLY)
            bbd_rtc_statistics(TRUE);
#else
            printf("} is only available on the Butterfly\n");
#endif
            break;
        case '<' :
        {
            VehicleID *veh_id = network_get_vehicle_id ();

            printf ("keyboard: Current <x,y,z> for vehicle num %d is\n",
                        veh_id -> vehicle );

            vec = kinematics_get_o_to_h (veh_kinematics);
            printf ("<%lf, %lf, %lf>\n", vec[0], vec[1], vec[2]);
        }
            break;
        case '.' :
            ammo_print_statistics ();
            printf ("ammo_print_statistics ()\n");
        case '~' :
            printf ("n_mapped = %d\n", get_n_mapped ());
            break;
        case ',' :
            print_reasons ();
            break;
        case '/' :
            printf ("cmdrs binoculars on\n");
            vision_cmdrs_binoculars (BINOC);
            break;
        case '\\' :
            if (get_ballistics_debug ())
            {
                set_ballistics_debug (FALSE);
                printf ("ballistics_debugging off\n");
            }
            else
            {
                set_ballistics_debug (TRUE);
                printf ("ballistics debugging on\n");
            }
            break;
        case '|':
            rtc_print_permanent();
            break;
        default :
            printf ("Unassigned character: type <?> for help\n");
            break;
    }
}

static void keyboard_setup_terminal ()
{
    console = keybrd_tty_init (0, O_RDONLY);
}

static void keyboard_reset_terminal ()
{
    keybrd_tty_reset (console);
}

void keyboard_exit_gracefully ()
{
    if (! use_keyboard)
    {
        return;
    }

    keyboard_reset_terminal ();
    keybrd_tty_close (console);
}
```

Appendix G:

Software to Control ESIG-500 Head Tracking Display

```
/*----------------------------------------------------------------------*/
/*                                                                      */
/*      HEAD MOUNTED DISPLAY - Image Generator Control Program          */
/*                                                                      */
/*                                                                      */
/*      FILENAME:  esighmd.cpp                                          */
/*                                                                      */
/*                                                                      */
/*      By:     - Visual Systems Laboratory                             */
/*              - Institute for Simulation and Training                 */
/*              - University of Central Florida                         */
/*                                                                      */
/*                                                                      */
/*      Copyright (c) 1991 the University of Central Florida            */
/*              - All Rights Reserved                                   */
/*                                                                      */
/*                                                                      */
/*      Author:         Richard Dunn-Roberts                            */
/*                                                                      */
/*                                                                      */
/*      FUNCTION LIST:                                                  */
/*      -------------                                                   */
/*                                                                      */
/*      FUNC: int initializeEthernet( int maxDataSize, int numPackets ) */
/*            This function initializes the Ethernet packet queues,     */
/*            the packet manager, and the 3Com Ethernet drivers.        */
/*                                                                      */
/*      General Comments:                                               */
/*              This program was written to run on a PC-AT, using       */
/*              Borland C++ version 2.0 (with the built-in assembler).  */
/*              It is designed to control the operation of the ESIG 500 */
/*              Image Generator for use with a Head-Mounted Display.    */
/*                                                                      */
/*                                                                      */
/*      Operational Comments:                                           */
/*              This program operates as a communications server.  It   */
/*              accepts input from the serial port.  This input may be  */
/*              from a Polhemus magnetic tracker or from another        */
/*              computer on a network.  The input is then converted to  */
/*              ESIG 500 Image Generator commands and retransmitted     */
/*              via dedicated Ethernet to the ESIG 500 to control       */
/*              the direction of the users point of view.               */
/*                                                                      */
/*              The basic layout is as follows:                         */
/*                                                                      */
/*      +-----------------+   19.2 kilobaud     +-----------------+     */
/*      |   I/O server    --------------------->|     PC-AT        |     */
/*      | (currently      |    serial link      |                 |     */
/*      |  Harris Night-  |                     |    (esighmd)    |     */
/*      |        Hawk)    |                     |                 |     */
/*      |                 |<--+                 +-------|---------+     */
/*      +-----------------+   |                         |               */
/*                            |                         |               */
/*           head position    |                         |               */
/*            and orientation |                         |               */
/*                            |                         |               */
/*      +-----------------+   |                         |               */
/*      |    POLHEMUS     |   |                         |               */
/*      |    TRACKER      ----+                         |               */
/*      |                 |                            \ /              */
/*      +-----------------+                     +-----------------+     */
/*                                              |                 |     */
/*                              video output    |    ESIG 500     |     */
/*                            +------------------      IG         |     */
/*                            |                 | (two channels)  |     */
/*                            |                 |                 |     */
/*                             |                +------------------+     */
/*                             |                                         */
/*      +------------------+   |                                         */
/*      |  CYBERFACE II    |   |                                         */
/*      |      HMD         |<--+                                         */
/*      |                  |                                             */
/*      +------------------+                                             */
/*                                                                       */
/*      For further system details, refer to the project report.         */
/*                                                                       */
/*-----------------------------------------------------------------------*/


/*---------------------------------*/
/* Type, Structure & Constant Defs */
/*---------------------------------*/

/*-------------------------*/
/* Necessary Include Files */
/*-------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <bios.h>
#include <time.h>
#include <dos.h>
#include <math.h>
#include <fstream.h>
#include "esighmd.h"
#include "esigcom.h"
#include "keypresl.h"
#include "serial.h"

/*---------------------*/
/* Function Prototypes */
/*---------------------*/

extern "C" int getopt( int argc, char far *argv[], char far *optionS );

/*---------*/
/* Globals */
/*---------*/

static unsigned long far *timer = (unsigned long far *)MK_FP(0,0x46C);
                                                         // BIOS timer

int numpackets = 0;   // number of packets that have been received

// Ethernet addresses of this machine (src) and ESIG 500 (dest)
// unsigned char src[] = { 0x02, 0x60, 0x8c, 0x43, 0xa7, 0xf1 };
// unsigned char dest[] = { 0x00, 0x08, 0x01, 0x57, 0x58, 0x00 };

// external globals for getopt
extern int     optind;         // index of which argument is next
extern char   *optarg;         // pointer to argument of current option
extern int     opterr;         // allow error message

/*-----------------------------------------------------------------------*/
/* Function main                                                         */
/*                                                                       */
/* PARAMETERS:                                                           */
/*   int argc - number of command line arguments                         */
/*   char **argv - array of string pointers to command line arguments    */
/*                                                                       */
/* PROCESS:                                                              */
/*    This is the main routine.                                         */
/*                                                                      */
/* RETURN VALUE:                                                        */
/*    int - unused by operating system, but available for user          */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
int main ( int argc, char **argv )
{
   int done = 0,             // completion flag is false to start
       retCode = 0,          // function success (0) or failure (!0)
       option,               // command line option
       printCount = 10000,   // how often do we print stats?
       syncErr = 0,          // number of serial sync errors to occur
       csNum;                // coordinate system number command goes to

   time_t startTime,         // timer values
          endTime;

   double xIn,               // intermediate value of x,y,z, pitch, heading,
          yIn,               //   roll received from serial port
          zIn,
          pitchIn,
          headingIn,
          rollIn;

   char syncChar,            // character to sync serial communications
        errFlag,             // did error occur in current cycle?
        csNumChar;           // character representing coord. system number

   long count = 0,           // number of cycles that have passed
        messageCount = 0,    // number of messages sent
        maxDataSize = 300,   // maximum size of ethernet data
        numPackets = 100,    // number of available ethernet packets
        viewX = 19,          // initial x, y, z, heading, pitch, roll
        viewY = 19,
        viewZ = 6,
        viewHeading = 180,
        viewPitch = 350,
        viewRoll = 0,
        x,                   // x, y, z, heading, pitch, roll used during
        y,                   // run
        z,
        heading,
        pitch,
        roll;

   // program takes 8 arguments, each with default values
   while( ( option = getopt( argc, argv, "d:n:x:y:z:p:h:" ) ) != EOF )
   {
      switch ( option )
      {
         case 'd' :
            maxDataSize = atoi( optarg );
            break;
         case 'n' :
            numPackets = atoi( optarg );
            break;
         case 'x' :
            viewX = atol( optarg );
            break;
         case 'y' :
            viewY = atol( optarg );
            break;
         case 'z' :
            viewZ = atol( optarg );
            break;
         case 'h' :
```

```
      viewHeading = atol( optarg );
      break;
    case 'p' :
      viewPitch = atol( optarg );
      break;
    case '?' :
      printf( "Invalid option, option ignored.\n" );
      break;
  }
}

// initialize the comport to receive commands
InitComPort( 0, DIVISOR(19200) );

// initialize the ethernet
retCode = initializeEthernet( maxDataSize, numPackets );
if ( retCode ) // something went wrong
{
  cprintf( "\n\rinitializeESIGControl() returns %d\n\r", retCode );
  return retCode;
}
else
{
  cprintf ( ">>> ESIG 500 Control Program\n\r" );
  cprintf ( ">>> UCF Institute for Simulation and Training\n\r" );
  cprintf ( "\n\r>>> Initialization complete\n\r" );
}

// initialize the viewpoint to the de position and orientation
//      NOTE: this is done twice or ESIG 500 takes it as delta value to
//            be applied at every time slice
retCode = escs( 0, 1, viewX*512, viewY*512, viewZ*512,
             viewHeading*182, viewPitch*182, 0L, messageCount++ );
if ( retCode ) // something went wrong
{
  cprintf( "\n\rescs() returns %d\n\r", retCode );
  return retCode;
}
// second time, see comment above
retCode = escs( 0, 1, viewX*512, viewY*512, viewZ*512,
           viewHeading*182, viewPitch*182, 0L, messageCount++ );
if ( retCode ) // something went wrong
{
  cprintf( "\n\rescs() returns %d\n\r", retCode );
  return retCode;
}

// since we want to know how fast things are going, start a timer
startTime = time( NULL );

while ( !done )
{

  // Reinit errFlag for the new cycle
  errFlag = 0;

  // Get the first character - is it the sync character?
  ReceiveData( 0, &syncChar, 1 );
  while ( syncChar != 's' )
  {
    if ( !errFlag )
    {
      syncErr++;
      errFlag = 1;
    }
    ReceiveData( 0, &syncChar, 1 );
```

```
}

// We are now synced up, proceed with processing

// First get the coordinate system number.  Currently this is always
// zero, but we must be general enough to control multiple coordinate
// systems
ReceiveData( 0, (char *)&csNumChar, 1 );

// convert the cs number to an int
csNum = csNumChar - '0';


// Next get the transmitted position and orientation values
// These are double floats
ReceiveData( 0, (char *)&xIn, 8 );
ReceiveData( 0, (char *)&yIn, 8 );
ReceiveData( 0, (char *)&zIn, 8 );
ReceiveData( 0, (char *)&headingIn, 8 );
ReceiveData( 0, (char *)&pitchIn, 8 );
ReceiveData( 0, (char *)&rollIn, 8 );

// here we adjust the heading of the viewpoint
headingIn += 180.0;
if ( headingIn > 360.0 )
   headingIn -= 360.0;

// Check cycle count.  If cycle count is evenly divisible by
// printCount, print a status message.
if ( !(count++ % printCount) )
   printf( "Cycle count = %ld\nbuffer contains %d, syncerrs %d, free packet n
           count, ReceiveBufferUsed( 0 ), syncErr, NumberOfFreePacketNodes()

// Convert the double floats into the long ints understood by the ESIG
x = ( long )floor( xIn * 512.0 );
y = ( long )floor( yIn * 512.0 );
z = ( long )floor( zIn * 512.0 );
heading = (long)floor(headingIn*182.0);
pitch = (long)floor(pitchIn*182.0);

// clamp the roll
// roll = (long)floor(rollIn*182.0);
roll = 0L;

// make new view point position relative to initial position
x += viewX*512;
y += viewY*512;
z += viewZ*512;

// transmit new viewpoint (twice, remember?)
retCode = escs(csNum, 1,  x, y, z, heading, pitch, roll, messageCount++);
if ( retCode ) // something went wrong
{
   cprintf( "\n\rescs() returns %d\n\r", retCode );
   return retCode;
}
retCode = escs(csNum, 1,  x, y, z, heading, pitch, roll, messageCount++);
if ( retCode ) // something went wrong
{
   cprintf( "\n\rescs() returns %d\n\r", retCode );
   return retCode;
}

// Poll the keyboard to see if the escape key has been hit
if ( KeyPressed() )
{
```

```
         unsigned key = bioskey(0);
         if ( key & 0xff ) key &= 0xff;
         if ( key == 0x1b ) // escape key
            done = 1;
      }

   }

   // get the end time
   endTime = time( NULL );

   // report time stats
   cprintf( "\n\rElapsed time is %f, cycleCount is %ld, frequency is %f\n\r",
            difftime( endTime, startTime ), messageCount,
            messageCount / difftime( endTime, startTime ) );

   // unload the serial driver from memory
   DeinstallDrivers();

   // exit program
   return 0;

}

/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* Function initializeEthernet                                          */
/*                                                                      */
/* PARAMETERS:                                                          */
/*    int maxDataSize - largest possible data packet to be sent         */
/*    int numPackets  - number of Ethernet packets to create            */
/*                                                                      */
/* PROCESS:                                                             */
/*    This function initializes the Ethernet packet queues, the packet  */
/*    manager, and the 3Com Ethernet drivers.                           */
/*                                                                      */
/* RETURN VALUE:                                                        */
/*    int - zero for success, nonzero for failure                       */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
int initializeEthernet( int maxDataSize, int numPackets )
{
   int retCode = 0;
   PacketNode *pnode;
   extern PacketQueue RxQueue;
   delay(1);                              /* Initialize delay() */
   retCode = InitPacketManager ( maxDataSize, numPackets );
   cWrRxFilter ( 0 );
   while ( !QueueEmpty( &RxQueue ) )
   {
      pnode = RemovePacket( &RxQueue );
      FreePacket( pnode );
   }

   return retCode;

}

/*--- end of file esighmd.cpp ---*/
```

```
#ifndef __cplusplus
#error  This program requires compilation as C++.
#endif

#ifndef __LARGE__
#error  This program requires Large memory model.
#endif

#ifndef __ESIGHMD
#define __ESIGHMD

// constants    ------------------------------------------------------------

// structs, classes, typedefs ----------------------------------------------

// macros and inline functions ---------------------------------------------

// function prototypes -----------------------------------------------------

/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* Function initializeEthernet                                            */
/*                                                                        */
/* PARAMETERS:                                                            */
/*    int maxDataSize - largest possible data packet to be sent           */
/*    int numPackets  - number of Ethernet packets to create              */
/*                                                                        */
/* PROCESS:                                                               */
/*    This function initializes the Ethernet packet queues, the packet  */
/*    manager, and the 3Com Ethernet drivers.                             */
/*                                                                        */
/* RETURN VALUE:                                                          */
/*    int - zero for success, nonzero for failure                         */
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
int initializeEthernet ( int maxDataSize, int numPackets );

#endif // __ESIGHMD
```

```
/*----------------------------------------------------------------------*/
/*                                                                      */
/*      HEAD MOUNTED DISPLAY - ESIG 500 IG Command Library              */
/*                                                                      */
/*                                                                      */
/*      FILENAME:  esigcom.cpp                                          */
/*                                                                      */
/*                                                                      */
/*      By:     - Visual Systems Laboratory                             */
/*              - Institute for Simulation and Training                 */
/*              - University of Central Florida                         */
/*                                                                      */
/*                                                                      */
/*      Copyright (c) 1991 the University of Central Florida            */
/*              - All Rights Reserved                                   */
/*                                                                      */
/*                                                                      */
/*      Author:         Richard Dunn-Roberts, Chuck Campbell            */
/*                                                                      */
/*                                                                      */
/*      FUNCTION LIST:                                                  */
/*      -------------                                                   */
/*                                                                      */
/*      FUNC: int esambient(int scene, int ambience, long messageCount) */
/*            This function sets ambient light levels.                  */
/*      FUNC: int esanimation( int cs, int select, int parcel,          */
/*                             int placeable, int control,              */
/*                             int sequence, long messageCount )        */
/*            This function executes a predefined animation sequence.   */
/*      FUNC: int eschannel( int number, int display_a, int display_b,  */
/*                         int viewport, int color, long messageCount ) */
/*            This function sets control parameters for specific        */
/*            channel on the ESIG 500.                                  */
/*      FUNC: int escloud( long top, long bottom, long messageCount )   */
/*            This function sets cloud top and bottom heights.          */
/*      FUNC: int escold( int color, int valid, int automatic,          */
/*                        long messageCount )                           */
/*            This functions sets characteristics of collision          */
/*            detection indicators                                      */
/*      FUNC: int escoldpt( int number, long x, long y, long z,         */
/*                          long messageCount )                         */
/*            This function sets collision detection test points        */
/*      FUNC: int escs( int csnum, int select, long x, long y, long z,  */
/*                      unsigned int heading, unsigned int pitch,       */
/*                      unsigned int roll, long messageCount )          */
/*            This functions controls characteristics of ig coordinate  */
/*            systems.                                                  */
/*      FUNC: int esdisable( int esig_switch, long messageCount )       */
/*      FUNC: int esenable( int esig_switch, long messageCount )        */
/*            These two functions enable and disable simulation         */
/*            characteristics, such as weather, strobes, and collision  */
/*            detection.                                                */
/*      FUNC: int esgfog( signed int top, long messageCount )           */
/*            This function sets the height of ground fog.              */
/*      FUNC: int eshatpt( int number, signed long x, signed long y,    */
/*                         signed long z, long messageCount )           */
/*            This function sets height above terrain test points.      */
/*      FUNC: int eshorizon( int brightness, int directional,           */
/*            unsigned int heading, long messageCount )                 */
/*            This function sets horizon brightness and direction.      */
/*      FUNC: int esinstructor( int channel, long messageCount )        */
/*            This function sets instructor monitor channel.            */
/*      FUNC: int eslight(int number, int intensity, long messageCount) */
/*            This function sets the intensity of light switches.       */
/*      FUNC: int eslobe( int lights, long messageCount )               */
/*            This function sets aircraft landing lights.               */
/*      FUNC: int esmodel(int colocate, int parcel, long messageCount)  */
/*            This function allows user to load a database.             */
/*      FUNC: int espolygon( int number, int intensity,                 */
/*                           long messageCount )                        */
/*            This function sets polygon intensity.                     */
/*      FUNC: int esrvr( long range, long messageCount )                */
/*            This function sets runway visibility range.               */
/*      FUNC: int esscene( int select, long messageCount )              */
/*            This function sets day, dusk, or night scene type.        */
/*      FUNC: int essun( unsigned int heading, unsigned int pitch,      */
/*                       long messageCount )                            */
/*            This function sets sun heading and pitch.                 */
/*      FUNC: int estraffic( int cs, int select, int parcel,            */
/*            int placeable, int control, int scenario,                 */
/*            long messageCount )                                       */
/*            This function sets information for routed or converging  */
/*            traffic.                                                  */
/*      FUNC: int esviewport( int channel, int alternate, long x,       */
/*            long y, long z, unsigned int heading, unsigned int pitch, */
/*            unsigned int roll, unsigned int vertical,                 */
/*            unsigned int horizontal, long messageCount )              */
/*            This function sets viewport characteristics.              */
/*      FUNC: int esvisibility( long range, long messageCount )         */
/*            This function sets visibility range.                      */
/*                                                                      */
/*      General Comments:                                               */
/*              This library was written to run on a PC-AT, using       */
/*              Borland C++ version 2.0.  It is designed to provide     */
/*              functional support to control the ESIG 500 image        */
/*              generator.                                              */
/*                                                                      */
/*                                                                      */
/*      Operational Comments:                                           */
/*              This library provides the interface to control the      */
/*              ESIG 500 from a user program.  The public functions     */
/*              take the user's inputs, converts them into an           */
/*              Ethernet packet, and retransmits the information to     */
/*              the ESIG 500.                                           */
/*                                                                      */
/*              The basic layout is as follows:                         */
/*                                                                      */
/*      +-----------------+   ethernet link     +-----------------+     */
/*      |     PC AT       --------------------->|     ESIG 500    |     */
/*      | (user program)  |                     |                 |     */
/*      |   (esigcom)     |                     |                 |     */
/*      |                 |                     |                 |     */
/*      |                 |                     +-----------------+     */
/*      +-----------------+                                             */
/*                                                                      */
/*      For further system details, refer to the project report.        */
/*                                                                      */
/*----------------------------------------------------------------------*/


/*-----------------------------------*/
/* Type, Structure & Constant Defs */
/*-----------------------------------*/

/*---------------------------*/
/* Necessary Include Files */
/*---------------------------*/
#include <stdlib.h>
#include "esigcom.h"

/*-----------------------*/
/* Function Prototypes */
```

```
/*=-=-=-=-=-=-=-=-=-=-=*/

/*=-=-=-=-=*/
/* Globals */
/*=-=-=-=-=*/

// Ethernet addresses of this machine (src) and ESIG 500 (dest)
unsigned char src[] = { 0x02, 0x60, 0x8c, 0x43, 0xa7, 0xf1 };
unsigned char dest[] = { 0x00, 0x08, 0x01, 0x57, 0x58, 0x00 };

/******************************************************************************/
/*  Function esambient                                                        */
/*                                                                            */
/*  PARAMETERS:                                                               */
/*    scene    - the scene whose brightness is being set.                     */
/*    ambience - the brightness value (0-255).                                */
/*    messageCount - sequence number of this message                          */
/*                                                                            */
/*  Process:                                                                  */
/*    Program to set brightness for the scene. Brightness is a scalar from    */
/*    0-255, and scene is 0 (night), 1 (dusk), or 2 (day).                    */
/*                                                                            */
/*  Returns:                                                                  */
/*    int - success == 0, failure != 0;                                       */
/*                                                                            */
/******************************************************************************/
int esambient( int scene, int ambience, long messageCount )
{

    struct ambient_struct *ambient;
    int retCode = 0;

    ambient = (struct ambient_struct *)calloc(1,sizeof(struct ambient_struct));
    ambient->hostmessage = messageCount;
    ambient->hostopcode = 0;
    ambient->ambience = ambience;
    ambient->endOfData = 0;

    switch( scene )
    {
        case 0:
                ambient->opcode = 0x0042;
                break;
        case 1:
                ambient->opcode = 0x0041;
                break;
        case 2:
                ambient->opcode = 0x0040;
                break;
        default: break;
    }

    retCode = TransmitPacket( ambient, sizeof( ambient_struct ), dest, sizeof( ambi

    free( ambient );

    return retCode;

}

/******************************************************************************/
/*  Function esanimation                                                      */
/*                                                                            */
/*  PARAMETERS:                                                               */
/*    cs        - the number of the coordinate system.                        */
/*    select    - the select switch.                                          */
/*    parcel    - the parcel index (0 - 255).                                 */
/*    placeable - 0 = local parcel, 1 = placeable parcel.                     */
/*    control   - animation control number.                                   */
/*    sequence  - animation sequence number (0 - 15).                         */
/*    messageCount - sequence number of this message                          */
/*                                                                            */
/*  Process:                                                                  */
/*    Executes a predefined animation sequence. Controls are:                 */
/*             ES_NO_INFO             - no effect.                            */
/*             ES_LOAD_ANIMATION      - Loads animation information           */
/*             ES_START_ANIMATION     - Starts animation                      */
/*             ES_STOP_ANIMATION      - Stops animation                       */
/*             ES_UNLOAD_ANIMATION    - Unloads animation                     */
/*                                                                            */
/*  Returns:                                                                  */
/*    int - success == 0, failure != 0;                                       */
/*                                                                            */
/******************************************************************************/
int esanimation( int cs, int select, int parcel, int placeable, int control,
                 int sequence, long messageCount )
{

    struct anim_struct *anim;
    int retCode = 0;

    anim = (struct anim_struct *)calloc(1,sizeof(struct anim_struct));
    anim->hostmessage = messageCount;
    anim->hostopcode = 0;
    anim->opcode = 0x001a;
    anim->cs = cs;
    anim->select = select;
    anim->parcel = parcel;
    anim->placeable = placeable;
    anim->xyz = 7;
    anim->control = (control<<4) + sequence;
    anim->endOfData = 0;

    retCode = TransmitPacket( anim, sizeof( anim_struct ), dest, sizeof( anim_struc

    free( anim );

    return retCode;
}

/******************************************************************************/
/*  Function eschannel                                                        */
/*                                                                            */
/*  PARAMETERS:                                                               */
/*    number    - the channel number to set.                                  */
/*    display_a - enable/disable display A.                                   */
/*    display_b - enable/disable display B.                                   */
/*    viewport  - main/alternate viewport select.                             */
/*    color     - main/alternate color select.                                */
/*    messageCount - sequence number of this message                          */
/*                                                                            */
/*  Process:                                                                  */
/*    Program to set information for a specific channel number (0-7).         */
/*    Corresponds to some of the ESIG CHANNEL commands.                       */
/*                                                                            */
/*  Returns:                                                                  */
/*    int - success == 0, failure != 0;                                       */
/*                                                                            */
/******************************************************************************/
int eschannel( int number, int display_a, int display_b,
                int viewport, int color, long messageCount )
{
    struct channel_struct *channel;
    int retCode = 0;

    channel = (struct channel_struct *)calloc(1,sizeof(struct channel_struct));
    channel->hostmessage = messageCount;
    channel->hostopcode = 0;
    channel->opcode = 0x0015;
    channel->number = number;
    channel->display_a = display_a;
    channel->display_b = display_b;
    channel->viewport = viewport;
    channel->color = color;
    channel->endOfData = 0;

    retCode = TransmitPacket( channel, sizeof( channel_struct ), dest, sizeof( chan

    free( channel );

    return retCode;
}

/*****************************************************************************/
/*  Function escloud                                                         */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    top      - value for height of top of clouds.                          */
/*    bottom - value for height of bottom of clouds.                         */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Program to set the cloud top and bottom (max. alt. is 24.8 mi.).       */
/*    Corresponds to the ESIG CLOUDS function.                               */
/*    Input has been scaled to correspond to the ESIG                        */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int escloud( long top, long bottom, long messageCount )
{
    struct cloud_struct *cloud;
    int retCode = 0;

    cloud = (struct cloud_struct *)calloc(1,sizeof(struct cloud_struct));
    cloud->hostmessage = messageCount;
    cloud->hostopcode = 0;
    cloud->top_opcode = 0x0017;
    cloud->top = top/4;
    cloud->bot_opcode = 0x0018;
    cloud->bottom = bottom/4;
    cloud->endOfData = 0;

    retCode = TransmitPacket( cloud, sizeof( cloud_struct ), dest, sizeof( cloud_st

    free( cloud );

    return retCode;
}

/*****************************************************************************/
/*  Function escold                                                          */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    color     - index into the color palette.                              */
/*    valid     - enable/disable color index validity.                       */
/*    automatic - enable/disable the automatic collision detection indicator.*/
/*    messageCount - sequence number of this message                           */
/*                                                                             */
/*                                                                             */
/*  Process:                                                                   */
/*    Program to set the characteristics of the collision detection indicator*/
/*    Corresponds to some of the ESIG COLD command options.                    */
/*                                                                             */
/*  Returns:                                                                   */
/*    int - success == 0, failure != 0;                                        */
/*                                                                             */
/*******************************************************************************/
int escold( int color, int valid, int automatic, long messageCount )
{
    struct cold_struct *cold;
    int retCode = 0, size = sizeof( cold_struct );

    cold = (struct cold_struct *)calloc(1,sizeof(struct cold_struct));
    cold->hostmessage = messageCount;
    cold->hostopcode = 0;
    cold->opcode = 0x0035;
    cold->color = color;
    cold->valid = valid;
    cold->automatic = automatic;
    cold->endOfData = 0;

    retCode = TransmitPacket( cold, size, dest, size );

    free( cold );

    return retCode;
}

/*******************************************************************************/
/*  Function escoldpt                                                          */
/*                                                                             */
/*  PARAMETERS:                                                                */
/*    number - testpoint number (0-31, 255 to clear all testpoints).           */
/*    x      - x offset.                                                       */
/*    y      - y offset.                                                       */
/*    z      - z offset.                                                       */
/*    messageCount - sequence number of this message                           */

/*                                                                             */
/*  Process:                                                                   */
/*    Program to implement collision detection test points.                    */
/*    Corresponds to the ESIG COLD function.                                   */
/*                                                                             */
/*  Returns:                                                                   */
/*    int - success == 0, failure != 0;                                        */
/*                                                                             */
/*******************************************************************************/
int escoldpt( int number, long x, long y, long z, long messageCount )
{
    struct coldpt_struct *coldpt;
    int *tmp, retCode = 0, size = sizeof( coldpt_struct );

    coldpt = (struct coldpt_struct *)calloc(1,sizeof(struct coldpt_struct));
    coldpt->hostmessage = messageCount;
    coldpt->hostopcode = 0;
    coldpt->opcode = 0x0033;
    coldpt->number = number;
    x *= 512; y *= 512; z *= 512;   /* scale x,y,z */
    tmp = (int *)&x;
    coldpt->high_x = tmp[1];
    coldpt->low_x  = tmp[0];
    tmp = (int *)&y;
    coldpt->high_y = tmp[1];
    coldpt->low_y  = tmp[0];
    tmp = (int *)&z;
    coldpt->high_z = tmp[1];
    coldpt->low_z  = tmp[0];
    coldpt->endOfData = 0;

    retCode = TransmitPacket( coldpt, size, dest, size );

    free( coldpt );

    return retCode;
}

/*****************************************************************************/
/*  Function escs                                                            */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    csnum   - the number of the coordinate system.                         */
/*    select  - the select switch.                                           */
/*    x       - x offset.                                                    */
/*    y       - y offset.                                                    */
/*    z       - z offset.                                                    */
/*    heading - heading angle.                                               */
/*    pitch   - pitch angle.                                                 */
/*    roll    - roll angle.                                                  */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Program to simulate most of the ESIG CS function in xyz mode           */
/*    x, y, z, heading, pitch, and roll must all be scaled so that when the  */
/*    user calls the function with x=100 the ESIG will set x=100.            */
/*    x,y, and z are longs, but must be byte swapped.  I broke these values  */
/*    into high and low words in order to get the function to work properly. */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int escs( int csnum, int select, long x, long y, long z, unsigned int heading,
          unsigned int pitch, unsigned int roll, long messageCount )
{

    int *tmp, retCode = 0;
    struct cs_struct *cs;

    cs = (struct cs_struct *)calloc(1,sizeof(struct cs_struct));
    cs->hostmessage = messageCount;
    cs->hostopcode = 0;
    cs->opcode = 0x001a;
    cs->cs = csnum;
    cs->select = select;
    cs->xyz = 7;                   /* use x,y,z mode                    */
    cs->control = 0;               /* no control information            */
    cs->el = 1;                    /* enable extrapolation              */
    cs->hpr = 7;                   /* enable heading, pitch, and roll   */
    tmp = ( int *) &x;             /* typecast into integer (word) size */
    cs->low_x = tmp[0];
    cs->high_x = tmp[1];
    tmp = ( int *) &y;
    cs->low_y = tmp[0];
    cs->high_y = tmp[1];
    tmp = ( int *) &z;
    cs->low_z = tmp[0];
    cs->high_z = tmp[1];
    cs->heading = heading;
    cs->pitch = pitch;
    cs->roll = roll;
    cs->endOfData = 0;

    retCode = TransmitPacket( cs, sizeof( cs_struct ), dest, sizeof( cs_struct ) );

    free( cs );

    return retCode;
}

/******************************************************************************/
/*  Function esdisable                                                        */
/*                                                                            */
/*  PARAMETERS:                                                               */
/*    esig_switch - the number of the switch to be disabled.                  */
/*    messageCount - sequence number of this message                          */
/*                                                                            */
/*  Process:                                                                  */
/*    Routine for disabling switches.                                         */
/*    Currently the switches are defined as follows:                          */
/*          1 -- storm                                                        */
/*          2 -- ground fog                                                   */
/*          3 -- patchy ground fog                                            */
/*          4 -- scudded clouds                                               */
/*          5 -- clouds                                                       */
/*          6 -- rain                                                         */
/*          7 -- lightning                                                    */
/*          8 -- light strings displayed with random/modeled intensity        */
/*          9 -- own-ship wing tip strobe                                     */
/*         10 -- own-ship anti-collision beacon                               */
/*         11 -- height above terrain                                         */
/*         12 -- collision detection                                          */
/*         13 -- collision detection indicator                                */
/*                                                                            */
/*  Returns:                                                                  */
/*    int - success == 0, failure != 0;                                       */
/*    nothing.                                                                */
/*                                                                            */
/******************************************************************************/
int esdisable( int esig_switch, long messageCount )
{
    int retCode = 0;

    switch( esig_switch )
    {
        case 1:
                retCode = esprocess_switch( 0x0001, messageCount );
                break;
        case 2:
                retCode = esprocess_switch( 0x0002, messageCount );
                break;
        case 3:
                retCode = esprocess_switch( 0x0003, messageCount );
                break;
        case 4:
                retCode = esprocess_switch( 0x0004, messageCount );
                break;
        case 5:
                retCode = esprocess_switch( 0x0005, messageCount );
                break;
        case 6:
                retCode = esprocess_switch( 0x0006, messageCount );
                break;
        case 7:
                retCode = esprocess_switch( 0x0007, messageCount );
                break;
        case 8:
                retCode = esprocess_switch( 0x0008, messageCount );
                break;
        case 9:
                retCode = esprocess_switch( 0x0009, messageCount );
                break;
        case 10:
                retCode = esprocess_switch( 0x000a, messageCount );
                break;
        case 11:
                retCode = esprocess_switch( 0x0030, messageCount );
                break;
        case 12:
                retCode = esprocess_switch( 0x0031, messageCount );
                break;
        case 13:
                retCode = esprocess_switch( 0x0034, messageCount );
                break;
        default: break;
    }

    return retCode;
}

/*****************************************************************************/
/*  Function esenable                                                        */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    esig_switch - the number of the switch to be enabled.                  */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Routine for enabling switches.                                         */
/*    Currently the switches are defined as follows:                         */
/*          1 -- storm                                                       */
/*          2 -- ground fog                                                  */
/*          3 -- patchy ground fog                                           */
/*          4 -- scudded clouds                                              */
/*          5 -- clouds                                                      */
/*          6 -- rain                                                        */
/*          7 -- lightning                                                   */
/*          8 -- light strings displayed with random/modeled intensity       */
/*          9 -- own-ship wing tip strobe                                    */
/*         10 -- own-ship anti-collision beacon                              */
/*         11 -- height above terrain                                        */
/*         12 -- collision detection                                         */
/*         13 -- collision detection indicator                               */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int esenable( int esig_switch, long messageCount )
{

    int retCode = 0;

    switch( esig_switch )
    {
        case 1:
                retCode = esprocess_switch( 0x8001, messageCount );
                break;
        case 2:
                retCode = esprocess_switch( 0x8002, messageCount );
                break;
        case 3:
                retCode = esprocess_switch( 0x8003, messageCount );
                break;
        case 4:
                retCode = esprocess_switch( 0x8004, messageCount );
                break;
        case 5:
                retCode = esprocess_switch( 0x8005, messageCount );
                break;
        case 6:
                retCode = esprocess_switch( 0x8006, messageCount );
                break;
        case 7:
                retCode = esprocess_switch( 0x8007, messageCount );
                break;
        case 8:
                retCode = esprocess_switch( 0x8008, messageCount );
                break;
        case 9:
                retCode = esprocess_switch( 0x8009, messageCount );
                break;
        case 10:
                retCode = esprocess_switch( 0x800a, messageCount );
                break;
        case 11:
                retCode = esprocess_switch( 0x8030, messageCount );
                break;
        case 12:
                retCode = esprocess_switch( 0x8031, messageCount );
                break;
        case 13:
                retCode = esprocess_switch( 0x8034, messageCount );
                break;
        default: break;
    }

    return retCode;
}

/******************************************************************************/
/*  Function esgfog                                                           */
/*                                                                            */
/*  PARAMETERS:                                                               */
/*    top - height of ground fog (max 6.2 mi.).                               */
/*    messageCount - sequence number of this message                          */
/*                                                                            */
/*  Process:                                                                  */
/*   Program to set the height of ground fog (max 6.2 mi.).                   */
/*                                                                            */
/*  Returns:                                                                  */
/*    int - success == 0, failure != 0;                                       */
/*                                                                            */
/******************************************************************************/
int esgfog( signed int top, long messageCount )
{
    struct gfog_struct *gfog;
    int retCode = 0, size = sizeof( gfog_struct );

    gfog = (struct gfog_struct *)calloc(1,sizeof(struct gfog_struct));
    gfog->hostmessage = messageCount;
    gfog->hostopcode = 0;
    gfog->opcode = 0x0044;
    gfog->top = top;
    gfog->endOfData = 0;

    retCode = TransmitPacket( gfog, size, dest, size );
    free( gfog );

    return retCode;
}

/*****************************************************************************/
/*  Function eshatpt                                                         */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    number - height above terrain testpoint number (0-31, 255 to clear).   */
/*    x      - x offset.                                                     */
/*    y      - y offset.                                                     */
/*    z      - z offset.                                                     */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Program to implement height above terrain test points.                 */
/*    Corresponds to the ESIG HAT function.                                  */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int eshatpt( int number, signed long x, signed long y, signed long z,
             long messageCount )
{
    struct hatpt_struct *hatpt;
    int retCode = 0, size = sizeof( hatpt_struct ), *tmp;

    hatpt = (struct hatpt_struct *)calloc(1,sizeof(struct hatpt_struct));
    hatpt->hostmessage = messageCount;
    hatpt->hostopcode = 0;
    hatpt->opcode = 0x0032;
    hatpt->number = number;
    x *= 512; y *= 512; z *= 512;   /* scale x,y,z */
    tmp = (int *)&x;
    hatpt->high_x = tmp[1];
    hatpt->low_x  = tmp[0];
    tmp = (int *)&y;
    hatpt->high_y = tmp[1];
    hatpt->low_y  = tmp[0];
    tmp = (int *)&z;
    hatpt->high_z = tmp[1];
    hatpt->low_z  = tmp[0];
    hatpt->endOfData = 0;

    retCode = TransmitPacket( hatpt, size, dest, size );

    free( hatpt );

    return retCode;
}

/*****************************************************************************/
/*  Function eshorizon                                                       */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    brightness  - horizon brightness (0-5).                                */
/*    directional - enable/disable horizon direction settable facility.      */
/*    heading     - horizon heading (0-360 degrees).                         */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Program to set horizon brightness and heading. Heading will be set if  */
/*    the directional flag is set.                                           */
/*    Corresponds to some of the ESIG SET commands.                          */
/*    Input for heading is scaled to perform as the ESIG function does       */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int eshorizon( int brightness, int directional, unsigned int heading,
               long messageCount )
{
    struct horizon_struct *horizon;
    int retCode = 0, size = sizeof( horizon_struct );

    horizon = (struct horizon_struct *)
                          calloc(1,sizeof(struct horizon_struct));
    horizon->hostmessage = messageCount;
    horizon->hostopcode = 0;
    horizon->opcode = 0x0011;
    horizon->directional = directional;
    horizon->brightness = brightness;
    horizon->heading = heading*182.04;
    horizon->endOfData = 0;

    retCode = TransmitPacket( horizon, size, dest, size );

    free( horizon );

    return retCode;
}

/*****************************************************************************/
/*  Function esinstructor                                                    */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    channel - port instructor monitor is on.                               */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Program to set the instructor monitor channel.                         */
/*    Corresponds to the ESIG SET function.                                  */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int esinstructor( int channel, long messageCount )
{
    struct instructor_struct *instructor;
    int retCode = 0, size = sizeof( instructor_struct );


    instructor = (struct instructor_struct *)
                          calloc(1,sizeof(struct instructor_struct));
    instructor->hostmessage = messageCount;
    instructor->hostopcode = 0;
    instructor->opcode = 0x0050;
    instructor->channel = channel;
    instructor->endOfData = 0;

    retCode = TransmitPacket( instructor, size, dest, size );

    free( instructor );

    return retCode;
}
```

```
/****************************************************************************/
/*  Function eslight                                                        */
/*                                                                          */
/*  PARAMETERS:                                                             */
/*    number    - light switch number (0-63).                               */
/*    intensity - intensity of light (0-5).                                 */
/*    messageCount - sequence number of this message                        */
/*                                                                          */
/*  Process:                                                                */
/*    Program to set the intensity of light switches.                       */
/*    Corresponds to some of the ESIG SWITCH functions.                     */
/*                                                                          */
/*  Returns:                                                                */
/*    int - success == 0, failure != 0;                                     */
/*                                                                          */
/****************************************************************************/
int eslight( int number, int intensity, long messageCount )
{
    struct light_struct *light;
    int retCode = 0, size = sizeof( light_struct );


    light = (struct light_struct *)calloc(1,sizeof(struct light_struct));
    light->hostmessage = messageCount;
    light->hostopcode = 0;
    light->opcode = 0x0012;
    light->number = number;
    light->intensity = intensity;
    light->endOfData = 0;

    retCode = TransmitPacket( light, size, dest, size );

    free( light );

    return retCode;
}

/****************************************************************************/
/*  Function eslobe                                                         */
/*                                                                          */
/*  PARAMETERS:                                                             */
/*    lights - number corresponding to which lights should be turned on.    */
/*             (0 = no lights, 127 = all).                                  */
/*    messageCount - sequence number of this message                        */
/*                                                                          */
/*  Process:                                                                */
/*    Program to set aircraft landing lights.                               */
/*    Corresponds to the ESIG LOBE function, although the user must give the */
/*    number corresponding to the bits he would set using the LOBE command.  */
/*                                                                          */
/*  Returns:                                                                */
/*    int - success == 0, failure != 0;                                     */
/*                                                                          */
/****************************************************************************/
int eslobe( int lights, long messageCount )
{
    struct lobe_struct *lobe;
    int retCode = 0, size = sizeof( lobe_struct );

    lobe = (struct lobe_struct *)calloc(1,sizeof(struct lobe_struct));
    lobe->hostmessage = messageCount;
    lobe->hostopcode = 0;
    lobe->opcode = 0x0013;
    lobe->lights = lights;
    lobe->endOfData = 0;
    retCode = TransmitPacket( lobe, size, dest, size );

    free( lobe );

    return retCode;
}

/****************************************************************************/
/*  Function esmodel                                                        */
/*                                                                          */
/*  PARAMETERS:                                                             */
/*    colocate - colocatable parcel select number to be used (0-7).         */
/*    parcel   - parcel number in the database to be used.                  */
/*    messageCount - sequence number of this message                        */
/*                                                                          */
/*  Process:                                                                */
/*    Program to allow user to load a database.                             */
/*    Corresponds to the ESIG MODEL function.                               */
/*                                                                          */
/*  Returns:                                                                */
/*    int - success == 0, failure != 0;                                     */
/*                                                                          */
/****************************************************************************/
int esmodel( int colocate, int parcel, long messageCount )
{
    struct model_struct *model;
    int retCode = 0, size = sizeof( model_struct );


    model = (struct model_struct *)calloc(1,sizeof(struct model_struct));
    model->hostmessage = messageCount;
    model->hostopcode = 0;
    model->opcode = 0x0014;
    model->colocate = colocate;
    model->parcel = parcel;
    model->endOfData = 0;

    retCode = TransmitPacket( model, size, dest, size );

    free( model );

    return retCode;
}

/****************************************************************************/
/*  Function espolygon                                                      */
/*                                                                          */
/*  PARAMETERS:                                                             */
/*    number    - polygon switch number (0-63).                             */
/*    intensity - intensity value (0-5).                                    */
/*    messageCount - sequence number of this message                        */
/*                                                                          */
/*  Process:                                                                */
/*    Program to set the polygon intensity.                                 */
/*    Corresponds to some of the ESIG SWITCH functions.                     */
/*                                                                          */
/*  Returns:                                                                */
/*    int - success == 0, failure != 0;                                     */
/*                                                                          */
/****************************************************************************/
int espolygon( int number, int intensity, long messageCount )
{
    struct polygon_struct *polygon;
    int retCode = 0, size = sizeof( polygon_struct );
    polygon = (struct polygon_struct *)
                calloc(1,sizeof(struct polygon_struct));
    polygon->hostmessage = messageCount;
    polygon->hostopcode = 0;
    polygon->opcode = 0x0019;
    polygon->number = number;
    polygon->intensity = intensity;
    polygon->endOfData = 0;

    retCode = TransmitPacket( polygon, size, dest, size );

    free( polygon );

    return retCode;
}

/******************************************************************************/
/*  Function esrvr                                                            */
/*                                                                            */
/*  PARAMETERS:                                                               */
/*    range - runway visual range in feet.                                    */
/*    messageCount - sequence number of this message                          */
/*                                                                            */
/*  Process:                                                                  */
/*    Program to set the runway visibility range.                             */
/*    Corresponds to the ESIG GFOG command. No scaling done yet.              */
/*    Input scaled to correspond to ESIG function.                            */
/*                                                                            */
/*  Returns:                                                                  */
/*    int - success == 0, failure != 0;                                       */
/*                                                                            */
/******************************************************************************/
int esrvr( long range, long messageCount )
{
    struct rvr_struct *rvr;
    int retCode = 0, size = sizeof( rvr_struct );


    rvr = (struct rvr_struct *)calloc(1,sizeof(struct rvr_struct));
    rvr->hostmessage = messageCount;
    rvr->hostopcode = 0;
    rvr->opcode = 0x0010;
    rvr->range  = range/4;
    rvr->endOfData = 0;

    retCode = TransmitPacket( rvr, size, dest, size );

    free( rvr );

    return retCode;
}

/******************************************************************************/
/*  Function esscene                                                          */
/*                                                                            */
/*  PARAMETERS:                                                               */
/*    select - the value of the scene type to be displayed.                   */
/*    messageCount - sequence number of this message                          */
/*                                                                            */
/*  Process:                                                                  */
/*    Scene is set with parameters corresponding to esig functions:           */
/*         0 = SET NIGHT                                                      */
/*         1 = SET DUSK                                                       */
/*         2 = SET DAY                                                        */
/*                                                                            */
/*  Returns:                                                                  */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int esscene( int select, long messageCount )
{
    struct scene_struct *scene;
    int retCode = 0, size = sizeof( scene_struct );


    scene = (struct scene_struct *)calloc(1,sizeof(struct scene_struct));
    scene->hostmessage = messageCount;
    scene->hostopcode = 0;
    scene->opcode = 0x0016;
    scene->select = select;
    scene->endOfData = 0;

    retCode = TransmitPacket( scene, size, dest, size );

    free( scene );

    return retCode;
}

/*****************************************************************************/
/*  Function essun                                                           */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    heading - heading angle for the sun.                                   */
/*    pitch   - pitch angle for the sun.                                     */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Program for setting the sun's heading and pitch.                       */
/*    Corresponds to ESIG functions SET SUNH and SET SUNP.                   */
/*    Input to this function must be scaled to correspond to ESIG function.  */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int essun( unsigned int heading, unsigned int pitch, long messageCount )
{
    struct sun_struct *sun;
    int retCode = 0, size = sizeof( sun_struct );


    sun = (struct sun_struct *)calloc(1,sizeof(struct sun_struct));
    sun->hostmessage = messageCount;
    sun->hostopcode = 0;
    sun->opcode = 0x0043;
    sun->heading = heading*182.04;
    sun->pitch = pitch*182.04;
    sun->endOfData = 0;

    retCode = TransmitPacket( sun, size, dest, size );

    free( sun );

    return retCode;
}

/*****************************************************************************/
/*  Function estraffic                                                       */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    cs        - the number of the coordinate system.                       */
/*    select    - the select switch.                                         */
/*    parcel    - the parcel index (0 - 255).                                */
/*    placeable - 0 = local parcel, 1 = placeable parcel.                    */
/*    control   - routed and converging control number.                      */
/*    scenario  - routed and converging scenario number (0 - 15).            */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Sets information for routed or converging traffic. Controls are:       */
/*             ES_NO_INFO               - no effect.                         */
/*             ES_LOAD_ROUTED           - Loads information for routed traffic */
/*             ES_START_ROUTED          - Starts routed traffic              */
/*             ES_STOP_ROUTED           - Stops routed traffic               */
/*             ES_UNLOAD_ROUTED         - Unloads routed traffic             */
/*             ES_LOAD_CONVERGING       - Load info for converging traffic   */
/*             ES_START_CONVERGING      - Start converging traffic           */
/*             ES_STOP_CONVERGING       - Stop converging traffic            */
/*             ES_UNLOAD_CONVERGING     - unload info for converging traffic */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int estraffic( int cs, int select, int parcel, int placeable, int control,
               int scenario, long messageCount )
{
    struct traffic_struct *traffic;
    int retCode = 0, size = sizeof( traffic_struct );


    traffic = (struct traffic_struct *)calloc(1,sizeof(struct traffic_struct));
    traffic->hostmessage = messageCount;
    traffic->hostopcode = 0;
    traffic->opcode = 0x001a;
    traffic->cs = cs;
    traffic->select = select;
    traffic->parcel = parcel;
    traffic->placeable = placeable;
    traffic->xyz = 7;
    traffic->control = (control<<4) + scenario;
    traffic->endOfData = 0;

    retCode = TransmitPacket( traffic, size, dest, size );

    free( traffic );

    return retCode;
}

/*****************************************************************************/
/*  Function esviewport                                                      */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    channel     - channel number (0-7).                                    */
/*    alternate   - enable/disable alternate view.                           */
/*    x           - x offset.                                                */
/*    y           - y offset.                                                */
/*    z           - z offset.                                                */
/*    heading     - heading orientation for the viewport (-360 - 360 deg.).  */
/*    pitch       - pitch orientation for the viewport (-360 - 360 deg.).    */
/*    roll        - roll orientation for the viewport (-360 - 360 deg.).     */
/*    vertical    - vertical half angle of viewport (0.5 - 90 deg.).         */
/*    horizontal - horizontal half angle of viewport (0.5 - 90 deg.).        */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Program to define the viewport, which defines how the image will appear*/
/*    on the display. Corresponds to some of the ESIG CHANNEL commands.      */
/*    Some inputs are scaled to correspond to the ESIG function.             */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int esviewport( int channel, int alternate, long x, long y, long z,
                unsigned int heading, unsigned int pitch, unsigned int roll,
                unsigned int vertical, unsigned int horizontal, long messageCount )
{
    struct viewport_struct *viewport;
    int *tmp;
    int retCode = 0, size = sizeof( viewport_struct );



    viewport = (struct viewport_struct *)
                    calloc(1,sizeof(struct viewport_struct));
    viewport->hostmessage = messageCount;
    viewport->hostopcode = 0;
    viewport->opcode = 0x001b;
    viewport->channel = channel;
    viewport->alternate = alternate;
    viewport->xyz = 7;              /* use x,y,z mode                      */
    viewport->el = 1;               /* enable extrapolation                */
    viewport->hpr = 7;              /* enable heading, pitch, and roll     */
    x *= 512; y *= 512; z *= 512;   /* scale x,y,z                         */
    tmp = ( int *) &x;              /* typecast into integer (word) size */
    viewport->low_x = tmp[0];
    viewport->high_x = tmp[1];
    tmp = ( int *) &y;
    viewport->low_y = tmp[0];
    viewport->high_y = tmp[1];
    tmp = ( int *) &z;
    viewport->low_z = tmp[0];
    viewport->high_z = tmp[1];
    viewport->heading = heading*182.04;
    viewport->pitch = pitch*182.04;
    viewport->roll = roll*182.04;
    viewport->vertical = vertical*182.04;
    viewport->horizontal = horizontal*182.04;
    viewport->endOfData = 0;

    retCode = TransmitPacket( viewport, size, dest, size );

    free( viewport );

    return retCode;
}

/*****************************************************************************/
/*  Function esvisibility                                                    */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    range - the range in feet of visibility (max 49.6 miles).              */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Program to simulate the ESIG VISIBILITY command.                       */
/*    Input scaled to correspond to ESIG function (in FEET).                 */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
```

```
/*****************************************************************************/
int esvisibility( long range, long messageCount )
{
    struct visibility_struct *visibility;
    int retCode = 0, size = sizeof( visibility_struct );


    visibility = (struct visibility_struct *)
                    calloc(1,sizeof(struct visibility_struct));
    visibility->hostmessage = messageCount;
    visibility->hostopcode = 0;
    visibility->opcode = 0x000F;
    visibility->range  = range/4;
    visibility->endOfData = 0;

    retCode = TransmitPacket( visibility, size, dest, size );

    free( visibility );

    return retCode;
}

/*****************************/
/* Object Functions: PRIVATE */
/*****************************/

/*****************************************************************************/
/*  Function esprocess_switch                                               */
/*                                                                           */
/*  PARAMETERS:                                                              */
/*    opcode - the opcode of the switch to be enabled/disabled.              */
/*    messageCount - sequence number of this message                         */
/*                                                                           */
/*  Process:                                                                 */
/*    Routine to build the enable/disable packet using information from the  */
/*    enable/disable routines.                                               */
/*                                                                           */
/*  Returns:                                                                 */
/*    int - success == 0, failure != 0;                                      */
/*                                                                           */
/*****************************************************************************/
int esprocess_switch( int opcode, long messageCount )
{
    struct switch_struct *esswitch;
    int retCode = 0, size = sizeof( switch_struct );


    esswitch = (struct switch_struct *)
                    calloc(1,sizeof(struct switch_struct));
    esswitch->hostmessage = messageCount;
    esswitch->hostopcode = 0;
    esswitch->opcode = opcode;
    esswitch->endOfData = 0;

    retCode = TransmitPacket( esswitch, size, dest, size );

    free( esswitch );

    return retCode;
}
```

```
// esigcom.h
#define NEED_PACKET_DEFINITIONS

#include "net.h"
#include "3com.h"
#include "packet.h"

#ifndef __cplusplus
#error  This program requires compilation as C++.
#endif

#ifndef __LARGE__
#error  This program requires Large memory model.
#endif

#ifndef __ESIGCOM
#define __ESIGCOM

// constants    ----------------------------------------------------------

// structs, classes, typedefs -------------------------------------------

struct ambient_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   int ambience;
   unsigned int endOfData;
};

struct anim_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned char select;
   unsigned char cs;
   unsigned char parcel;
   unsigned char placeable;
   unsigned char xyz;
   unsigned char control;
   int dummy1;   /* dummys are introduced to give the structure */
   int dummy2;   /* the proper length of fourteen words         */
   int dummy3;
   int dummy4;
   int dummy5;
   int dummy6;
   int dummy7;
   int dummy8;
   int dummy9;
   int dummy10;
   unsigned int endOfData;
};

struct channel_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   char dummy;
   char number;
   char display_b;
   char display_a;
   char color;
   char viewport;
   unsigned int endOfData;
};

struct cloud_struct
{
   long hostmessage;
   int hostopcode;
   int top_opcode;
   signed int top;
   int bot_opcode;
   signed int bottom;
   unsigned int endOfData;
};

struct cold_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   int color;
   char automatic;
   char valid;
   unsigned int endOfData;
};

struct coldpt_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned char number;
   char dummy;
   int high_x;
   int low_x;
   int high_y;
   int low_y;
   int high_z;
   int low_z;
   unsigned int endOfData;
};

struct cs_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned char select;
   unsigned char cs;
   unsigned char index;
   unsigned char type;
   unsigned char xyz;
   unsigned char control;
   unsigned char el;
   unsigned char hpr;
   int high_x;
   int low_x;
   int high_y;
   int low_y;
   int high_z;
   int low_z;
   unsigned int heading;
   unsigned int pitch;
   unsigned int roll;
   unsigned int endOfData;

};
```

```
struct switch_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned int endOfData;
};

struct gfog_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   signed int top;
   unsigned int endOfData;
};

struct hatpt_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   char number;
   char dummy;
   int high_x;
   int low_x;
   int high_y;
   int low_y;
   int high_z;
   int low_z;
   unsigned int endOfData;
};

struct horizon_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   char directional;
   char brightness;
   unsigned int heading;
   unsigned int endOfData;
};

struct instructor_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   char channel;
   char dummy;
   unsigned int endOfData;
};

struct light_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   char intensity;
   char number;
   unsigned int endOfData;
};

struct lobe_struct
```

```
{
   long hostmessage;
   int hostopcode;
   int opcode;
   char lights;
   char dummy;
   unsigned int endOfData;
};

struct model_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   char parcel;
   char colocate;
   unsigned int endOfData;
};

struct polygon_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   char intensity;
   char number;
   unsigned int endOfData;
};

struct rvr_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned int range;
   unsigned int endOfData;
};

struct scene_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned char select;
   unsigned char dummy;  /* used for word alignment */
   unsigned int endOfData;
};

struct sun_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned int heading;
   unsigned int pitch;
   unsigned int endOfData;
};

struct traffic_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned char select;
   unsigned char cs;
   unsigned char parcel;
   unsigned char placeable;
   unsigned char xyz;
   unsigned char control;
   int dummy1;   /* dummys are introduced to give the structure */
   int dummy2;   /* the proper length of fourteen words         */
   int dummy3;
   int dummy4;
   int dummy5;
   int dummy6;
   int dummy7;
   int dummy8;
   int dummy9;
   int dummy10;
   unsigned int endOfData;
};

struct viewport_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned char alternate;
   unsigned char channel;
   unsigned char xyz;
   unsigned char dummy;
   unsigned char el;
   unsigned char hpr;
   int high_x;
   int low_x;
   int high_y;
   int low_y;
   int high_z;
   int low_z;
   unsigned int heading;
   unsigned int pitch;
   unsigned int roll;
   unsigned int vertical;
   unsigned int horizontal;
   unsigned int endOfData;
};

struct visibility_struct
{
   long hostmessage;
   int hostopcode;
   int opcode;
   unsigned int range;
   unsigned int endOfData;
};

// macros and inline functions -------------------------------------------

/*=-=-=-=-=-=-=-=-=-=-=*/
/* Function Prototypes */
/*=-=-=-=-=-=-=-=-=-=-=*/

int esambient( int scene, int ambience, long messageCount );
int esanimation( int cs, int select, int parcel, int placeable, int control,
                 int sequence, long messageCount );
int eschannel( int number, int display_a, int display_b,
                int viewport, int color, long messageCount );
int escloud( long top, long bottom, long messageCount );
int escold( int color, int valid, int automatic, long messageCount );
int escoldpt( int number, long x, long y, long z, long messageCount );
int escs( int csnum, int select, long x, long y, long z, unsigned int heading,
          unsigned int pitch, unsigned int roll, long messageCount );
```

```
int esdisable( int esig_switch, long messageCount );
int esenable( int esig_switch, long messageCount );
int esgfog( signed int top, long messageCount );
int eshatpt( int number, signed long x, signed long y, signed long z,
             long messageCount );
int eshorizon( int brightness, int directional, unsigned int heading,
               long messageCount );
int esinstructor( int channel, long messageCount );
int eslight( int number, int intensity, long messageCount );
int eslobe( int lights, long messageCount );
int esmodel( int colocate, int parcel, long messageCount );
int espolygon( int number, int intensity, long messageCount );
int esrvr( long range, long messageCount );
int esscene( int select, long messageCount );
int essun( unsigned int heading, unsigned int pitch, long messageCount );
int estraffic( int cs, int select, int parcel, int placeable, int control,
               int scenario, long messageCount );
int esviewport( int channel, int alternate, long x, long y, long z,
                unsigned int heading, unsigned int pitch, unsigned int roll,
                unsigned int vertical, unsigned int horizontal, long messageCount )
int esvisibility( long range, long messageCount );
int esprocess_switch( int opcode, long messageCount );


// variable externs ---------------------------------------------------

extern int maxTxLength;         // Maximum packet length transmitted

#endif // __ESIGCOM
```

Appendix H:

Summary of Results of HTD Experiments

Head-Tracked Cupola Display Preliminary Results

The results of this evaluation with novice subjects indicated that there were no significant differences in performance or preferences between the two simulations (Simnet vs HTD). It appears that for the selected tasks (target acquisition and navigation), subjects performed equally well (or equally poorly) in both simulations. There are several potential explanations for these findings: 1) the two simulations may be close enough in design that performance differences will not be seen for most tasks, 2) the tasks were simply too easy to determine if there were performance differences, and 3) there were too many problems with experimental control (e.g., too many disturbances, equipment failures, etc.) to keep variability low enough to observe any differences.

In my opinion, explanations one and two are probably both correct. From what I have observed and heard about tank scenarios, in most cases there probably would be no differences in performance between the two simulations. For example, the navigation task required infrequent head turning or cupola rotation to find a checkpoint. The tank commander (TC) was only required to tell the driver to turn, and the TC could easily determine his surroundings while looking straight ahead. I believe that the results from most navigation tasks would be similar. Concerning target acquisition, I have heard that in most situations the tank would be in a defensive position, or it would attack in groups, with each group focusing on a specific sector. In these situations, speed of acquisition would likely be less important. If this is true, then there would be little expectation for differences in performance between the two simulations, which was observed in this study.

However, if speed of target acquisition is important, then I predict that the HTD would provide superior performance. The target acquisition task in this study appeared to be too easy to elicit performance differences between the simulations, even though the 20 targets were placed in difficult locations (hence, the low acquisition mean of 11.75).

I also believe that the lack of experimental control concerning demos, equipment failures, etc., was a problem. However, these factors probably occurred equally over the experimental sessions, so I doubt if they influenced the means, except possibly to reduce them across the board. These problems could have increased the variability, which would require greater differences between the simulations to show an effect. In addition, a larger sample size was needed in order to increase the power to detect group differences.

In conclusion, I would recommend continuing the evaluation based on the importance of target acquisition speed versus the cost/benefits of continuing. If tank training does not focus on speed of target acquisition, then the Simnet is probably adequate for training, and continuing the study would not be cost effective. However, if speed of acquisition is very important, then we probably should continue with experienced personnel.

If we do continue, I have some further recommendations. We should drop the navigation task, because I doubt if there are any scenarios in which the two simulations would show differences. The navigation task also required more time to complete than the target acquisition task. Dropping the navigation task would allow us to run more subjects in a shorter amount of time.

As you suggested, Ernie, I would run the target acquisition task over a shorter route but with a much higher target density. This would require that the TC scan his environment faster, which may result in performance differences between the HTD and Simnet. I would also have the gunner-tank commander relationship more involved by having the gunner actually try to shoot the targets.

Finally, we need to run this experiment with better control. I would like to have access to two students who are available to run the experiment at any time, not just when they can fit it into their schedules, and I want to be able to limit people from interrupting the study. I realize that these requests may be difficult to accomplish.

Preliminary Results - Terrain Reasoning (N=12)
(accuracy between checkpoints 1-2)

MEDIANS BY SIMULATION AND COURSE

SIM/COURSE 1	SIM/COURSE 2	HTD/COURSE 1	HTD/COURSE 2
1 m	200 m	1 m	50.5
(R: 1-400)	(R: 1-700)	(R: 1-1600)	(R: 1-300)
Mean: 67.5	Mean: 266.8	Mean: 284.0	Mean: 100.5

MEDIANS BY SIMULATION

SIMNET	HTD
50.5 m	1 m
(R: 1-700)	(R: 1-1600)
Mean: 167.2	Mean: 192.3

MEDIANS BY COURSE

COURSE 1	COURSE 2
1 m	150 m
(R: 1-1600)	(R: 1-700)
Mean: 175.8	Mean: 183.7

Preliminary Results - Terrain Reasoning (N=12)
(time between checkpoints 2-3)

MEDIANS BY SIMULATION AND COURSE

SIM/COURSE 1	SIM/COURSE 2	HTD/COURSE 1	HTD/COURSE 2
253 sec. (R: 206-999)	268.5 sec. (R: 209-338)	238 sec. (R: 173-999)	221.5 sec. (R: 180-312)

MEDIANS BY SIMULATION

SIMNET	HTD
268.5 (R: 206-999)	225.5 (R: 173-999)

MEDIANS BY COURSE

COURSE 1	COURSE 2
238 (R: 173-999)	233.5 (R: 180-338)

Preliminary Results - Terrain Reasoning (N=12)
(accuracy between checkpoints 2-3)

MEDIANS BY SIMULATION AND COURSE

SIM/COURSE 1	SIM/COURSE 2	HTD/COURSE 1	HTD/COURSE 2
600 m (R: 1-9999)	50.5 m (R: 1-200)	650 m (R: 1-9999)	550 m (R: 200-1700)

MEDIANS BY SIMULATION

SIMNET	HTD
150 m (R: 1-9999)	550 m (R: 1-9999)

MEDIANS BY COURSE

COURSE 1	COURSE 2
600 m (R: 1-9999)	200 m (R: 1-1700)

Preliminary Results - Target Acquisition Means (N=12) (20 Targets were possible)

MEANS BY SIMULATION AND COURSE

SIM/COURSE 1	SIM/COURSE 2	HTD/COURSE 1	HTD/COURSE 2
12.50 (R: 10-14)	11.00 (R: 8-15)	12.67 (R: 9-17)	10.83 (R: 8-15)

MEANS BY SIMULATION

SIMNET	HTD
11.75 (R: 8-15)	11.75 (R: 8-17)

MEANS BY COURSE

COURSE 1	COURSE 2
12.59 (R: 9-17)	10.92 (R: 8-15)

Head-Tracked Cupola Display Research

Preliminary Results - Preference Means (N=13)

Scale: 5=impossible, 4=difficult, 3=moderate, 2=easy, 1=very easy

QUESTION	SIMNET	HTD
1.) Ability to perceive locations accurately	3.11	2.89
2.) Ability to identify surroundings	3.00	3.00
3.) Ability to maintain orientation and not get lost	3.0	2.67
4.) Ability to acquire targets without time constraints	2.67	2.67
5.) Ability to acquire targets under time pressure	3.44	3.00

None of these results were statistically significant.

6.) Which simulation was more difficult?

SIMNET=	6
HTD=	5
SAME=	2

7.) Were there any features about either simulation that you especially liked or disliked?

8.) Which simulation do you think would be the most beneficial for training?

SIMNET=	8
HTD=	4
SAME=	1

9.) Which simulation do you prefer?

SIMNET=	7
HTD=	5
SAME=	1

10.) Do you think that your performance and preferences would have changed if the head-tracked display did not have popping/flickering?

Yes = 11 No = 2

Appendix I

Computations Concerning EyePhone Resolution

The low resolution Eyephone has 442 x 238 primary color pixels per eye. The total number of primary color pixels is 105,196. This means 35,065 triads, in an array that I am assuming is 256 x 137. The FOV for each eye is 86 degrees (horizontal) by 76 degrees (vertical). The total horizontal FOV is 108 degrees. The binocular overlap is 64 degrees.

Because of assumptions made above, the following calculations are approximate. The total number of pixels (three color) horizontally is 256 x 108 / 86 (pixels per eye x total FOV / FOV per eye). This equals 321 pixels. The number of arc minutes per pixel horizontally is 108 x 60 / 321 or approximately 20 arc minutes per pixel.

The number of vertical three color pixels is approximately 137. The number of vertical degrees is 76. This gives approximately 33 arc minutes per pixel vertically.

The same calculations, with the same assumptions, applied to the high resolution Eyephones gives the following results.

Total number of primary color pixels per eye is 345,600 in a 720 by 480 matrix. This gives about 416 horizontal by 277 vertical three color pixels per eye. The horizontal FOV per eye is 86 degrees, and for both eyes is 106 degrees. The vertical FOV is 75 degrees. The total number of horizontal pixels is approximately 513 pixels. This works out to about 12 arc minutes per pixel horizontally. The number of three color pixels vertically is about 277. This works out to about 16 arc minutes per pixel vertically.

Appendix J:

Extending the SIMNET Head-Tracking Display

Extending the SIMNET Head Tracking Display: a Project Analysis by IST/VSL

J. Michael Moshell
Curtis Lisle
Richard Dunn-Roberts
Ernie Smart

VSL Memo 91.6
2/15/91

There is significant interest in extending the functionality of the SIMNET M1 trainer to include the Protected Open Position for the tank commander (referred to as the "POP hatch"). This document includes the results of a preliminary analysis of this problem.

The major sections of the document are as follows:

I. Project Requirements and Description
II. Constraints on the Image Generator and Monitors
III. Miscellaneous Concerns about the Design
IV. IST Estimates on Time and Cost

I. Project Requirements and Description

List of Project Requirements: The following list covers the main points of theproposed design. This serves as a summary of the overall project goals.

1. Provide a 360° field of view POP hatch display.

2. Allow the commander the maximum possible vertical field of view in the POP hatch display, consistent with tank geometry.

3. The POP hatch view will adjust for head motion within the cupola, so as to provide a high resolution central display, medium resolution lateral displays, and no imagery outside of the central viewing cone. This will conserve image generator channel capacity.

4. Visible elements of the tank (tank hull, main gun,etc) will be visible in the views from the POP hatch.

5. The cupola (including vision blocks, machine gun and hatch cover) will rotate under the control of the machine gun control handle.

6. The capability will be provided of using the 6 vision blocks as well as the POP hatch views simultaneously.

Description of the Requirements: The following paragraphs provide more detail about each of the project requirements stated above.

1. *Provide a 360° horizontal field of view.* The display will be created by providing a ring of ten 27" (diagonal) video monitors surrounding the tank commander's cupola. This will be an extension of the Head Tracking Display project (HTD) currently underway at IST/VSL.

Referred to as the Extended Head Tracking Display (EHTD), this new project will use the same video switching technique employed in IST's HTD: a Polhemus magnetic tracker is used to switch video channels so the commander always has an active video display in the direction his head is facing. (See Figure 1).

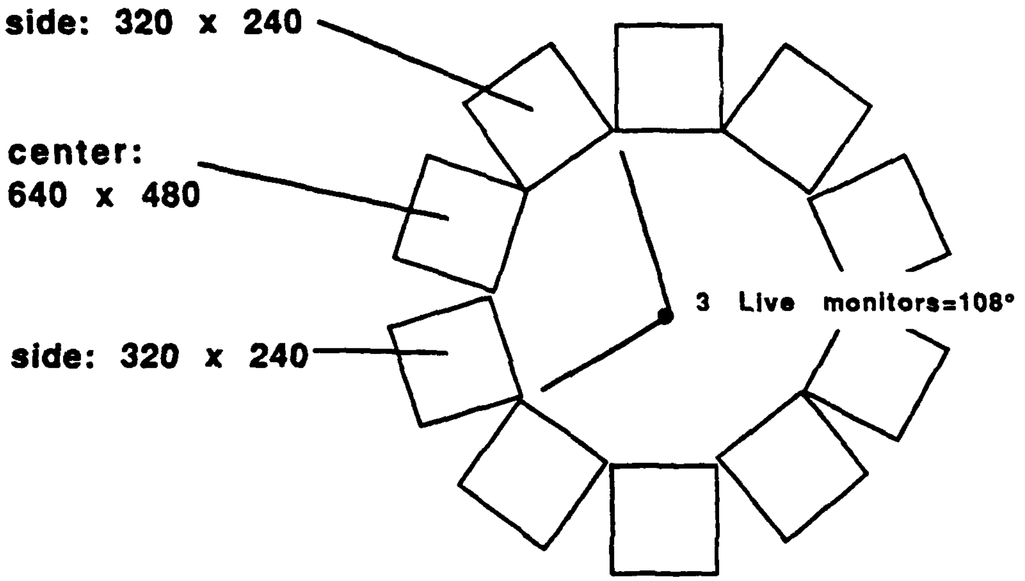


Figure 1 - Top View of the EHTD

2. *The commander is given the maximum feasible FOV in the vertical direction*: This requirement is based on the 6" high opening available under the elevated hatch and the commander's head position relative to the radius of the hatch. The geometry involved is shown in Figure 2.

> *Note: the aspect ratio of off-the-shelf NTSC monitors makes it limits the vertical field of view which can be achieved with a single row of monitors and no optics. See section II for details.*

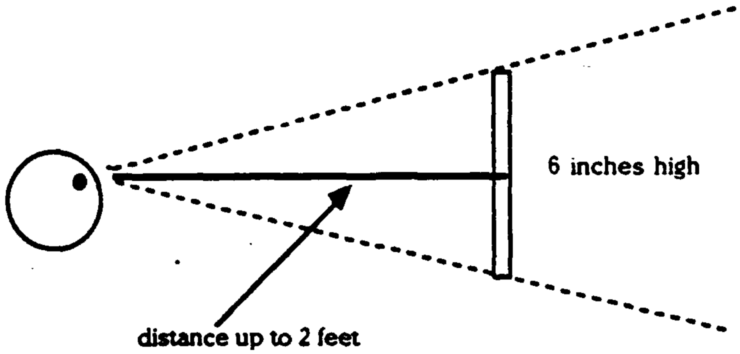


Figure 2 - Vertical field of view

3. *The POP hatch view will adjust for head motion within the cupola:* Since the commander has a range of available head motion within the cupola (approx. 32" in diameter in SIMNET), the view out the monitors should adjust for the correct head position. This requires head position information be passed over to the SIMNET Host and used to adjust the viewpoint generated by the SIMNET CIG unit. Engineering development will be necessary to provide this level of control over the CIG (greater than in the existing SIMNET cupola).

> *An additional concern is the obstacle provided by the vertical edges of the monitor housings. This is discussed in section III below.*

4. *Visible elements of the tank (fenders, gun,etc) will be displayed in the views from the POP hatch:* Since the commander's eyepoint will be less than 6" above the top of the turret in the POP hatch position, portions of the tank will often be visible. The visible tank features must be modelled so they will be displayed correctly.

5. *The cupola will rotate.* The current SIMNET simulator allows the commander's cupola to mechanically rotate but provides only a restricted view out of one vision block of the cupola. In a real tank, the cupola is sometimes rotated so as to position the machine gun out of the forward line of sight, or to aim the machine gun.

The EHTD project will support cupola rotation. Using an existing SIMNET hull and cupola mechanism, the required vision blocks and a simulated hatch cover and machine gun mount will be incorporated so as to rotate within the panoramic monitor display.

6. *Provide the capability of using the 6 vision blocks as well as the POP hatch views simultaneously:* The commander will be able to support training in normal mode (through the 6 vision blocks) or POP hatch (through the out the window monitors). Two competing designs are being considered to support this:

1. Have a set of monitors dedicated to the vision blocks and a set dedicated to the POP hatch display. A side view of this is shown in Figure 3. The monitors would have to occupy a carousel which rotates along with the cupola and vision blocks.

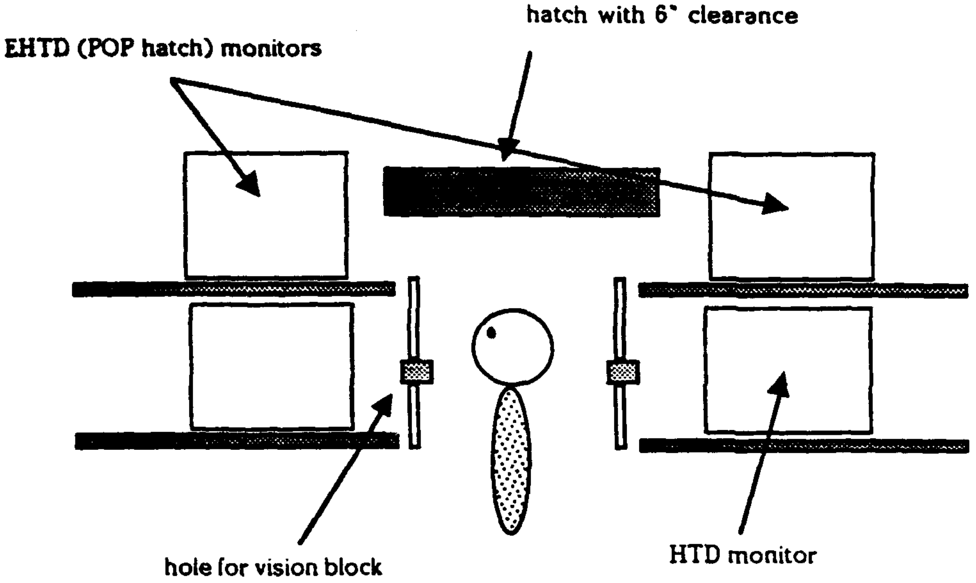


Figure 3 - Dual Set of Monitors

2. Use a single set of monitors to support both the vision blocks and the POP hatch display. This requires actual "periscope-style" vision blocks and a redesign of the approach used to supply the vision block views. This approach is shown in Figure 4.

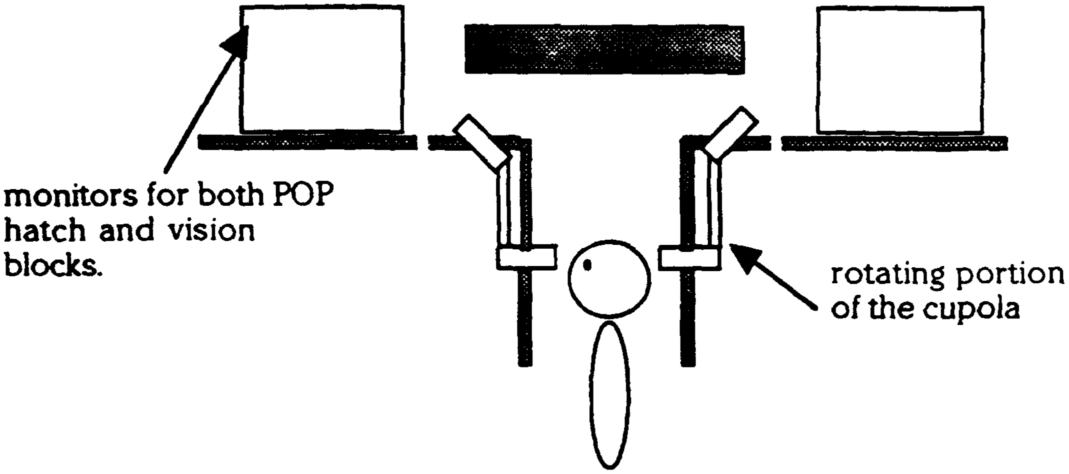

monitors for both POP hatch and vision blocks.
rotating portion of the cupola

Figure 4 - Single Set of Monitors

IV. IST Estimates on Time and Cost

Tha following spreadsheet titled *EHTD Equipment Pricing* shows estimated costs for materials for a one-of-a-kind installation. These would diminish by 10 to 20% in a production operation for 10 to 50 units.

The equipment estimate would need to be increased by the (unknown) amount which a contractor would charge for integrating the panoramic display into a production-model SIMNET tank hull for actual training purposes.

The spreadsheet titled *EHTD Personnel Pricing* shows IST's projected labor and overhead cost for performing the construction of a proof-of-concept demonstration. We have no way to estimate the charges a for-profit contractor would levy for similar services.

We estimate that six months would be required from receipt of a commitment to completion and testing of the EHTD system. The principal delays are involved with the acquisition of the IG and the debugging and testing of the host software, some of which would be developed in collaboration with a subcontractor.

This is an estimate, not a formal quotation, due to the haste with which the estimate was required.

EHTD Personnel Pricing

Labor Pricing for the EHTD Project

Category				Loaded:
Proj Mgr	Lisle	6 months	50%	16705
Vis Scientist	Dunn-Roberts	6 months	50%	14796
Engineer	to hire	3 months	50%	18604
Software sp.	to hire	3 months	50%	11455
elect constr	student	6 months	50%	7814
cptr prog	student	3 months	50%	3907
host progr.	PRC subcon.			12739
mech design	student	3 months	25%	1952
mech constr	student	3 months	25%	1952
consulting				10000
travel				8000
supplies				520
				108444